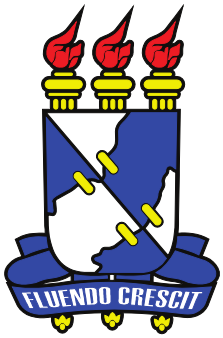


UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Desenvolvimento de um framework para a plataforma FIWARE

Trabalho de Conclusão de Curso

Romário Bispo da Silva

São Cristóvão – Sergipe

2019

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Romário Bispo da Silva

Desenvolvimento de um framework para a plataforma FIWARE

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador(a): Giovanny Fernando Lucero Palma

São Cristóvão – Sergipe

2019

Dedico essa tese à toda minha família, amigos e professores que me deram o conhecimento necessário para tal.

Agradecimentos

Gostaria de agradecer primeiramente a Deus, que ao longo desse jornada se mostrou ainda mais importante em minha vida, por toda força e motivação que me deu e que tem me dado. Agradeço por sempre me ouvir quando precisava e por me guiar pelo caminho correto que me levou até aqui, ao meu amadurecimento pessoal e profissional.

Agradeço também à minha família, por todo esforço, apoio, amor e dedicação. Em especial à minha mãe, que além disso, me aconselhou e fez de tudo por mim mesmo que de longe. Agradeço por todos os momentos em família que me moldaram como cidadão e como homem.

Agradeço aos meus professores que me instruíram durante toda esta jornada, com orientações técnicas e conselhos para a vida. Agradeço ao meu orientador Giovanny Lucero, que acreditou em mim, e que com suas orientações e conselhos, fez com que este trabalho se tornasse possível.

Agradeço à todos com que tive o prazer de trabalhar no STI, mesmo com apenas pouco mais de um ano de "casa", sempre me senti muito bem acolhido desde o primeiro dia, atribuo grande parte do meu amadurecimento profissional à todos do STI, foi o período que mais aprendi durante a minha graduação, bem como o mais divertido. Entre todos os profissionais do STI, tive a grande satisfação de ser supervisionado por Luís Souza, o Scrum Maestro. Agradeço a ele por toda a confiança e respaldo que depositou em mim, pelos conselhos e recomendações. Muito Obrigado!

Agora, chegou a parte mais difícil, como listar todos os amigos que fiz ? na verdade, isso me tomaria ainda mais páginas. No entanto, quero destacar alguns dos meus grandes amigos que compartilharam desta árdua jornada na graduação, como o meu amigo Hector Hiroshi, que conheci no dia da matrícula e que compartilhamos das dificuldades como colegas em quase todos os trabalhos em grupo e que mesmo sofrendo, tirávamos motivo pra sorrir. Agradeço ao meu amigo de infância Gilcley que sempre me demonstrou grande apreço e que ao longo dos anos se tornou para mim um exemplo como pessoa e profissional. Quero agradecer também aos meus colegas de residência, em especial, ao meu amigo David Costa, pelas conversas, risadas e conselhos furados. Aos que não pude citar aqui, saibam que são tão importantes quanto. Obrigado!

Sonhos determinam o que você quer.
Ação determina o que você conquista.
(Aldo Novak)

Resumo

A plataforma FIWARE tem se mostrado promissora para o desenvolvimento de aplicações para internet das coisas e cidades inteligentes. Porém, a velocidade de aprendizado das ferramentas da plataforma é lenta e suas APIs REST dificultam a obtenção de código conciso e modular. Este trabalho propõe um protótipo de um *framework* de programação Java que tem a plataforma FIWARE como *backend*. O protótipo abstrai a montagem de requisições HTTP assim como também a serialização/deserialização de objetos no formato JSON. Ela se apresenta como uma abordagem pura de programação orientada a objetos, contribuindo para um código mais claro e modular. Por fim, aplicações utilizadas como estudo de caso em outros trabalhos foram aqui reescritas utilizando o *framework*. Estas implementações mostram que efetivamente há melhora da qualidade do código, refletindo na sua modularidade.

Palavras-chave: Internet das coisas, Cidades Inteligentes, FIWARE.

Abstract

The FIWARE platform has shown promise for developing IoT applications and smart cities. However, the platform's learning speed is slow and its REST APIs make it difficult to obtain concise and modular code. This work proposes a prototype of a Java programming framework that has the FIWARE platform as backend. The prototype abstracts the assembly of HTTP requests as well as serialization / deserialization of objects in JSON format. It presents itself as a pure object-oriented approach to programming, contributing to clearer, more modular code. Finally, applications used as a case study in other works were rewritten here using the framework. These implementations show that code quality is effectively improved, reflecting its modularity.

Keywords: Internet of Things, Smart Cities, FIWARE.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CB	Context Broker
CE	European Commission
CKAN	Comprehensive Knowledge Archive Network
CRUD	Create, Read, Update and Delete
CoAP	Constraint Application Protocol
GEs	Generic Enablers
HDFS	Hadoop Distributed File System
HTTP	HyperText Transfer Protocol
TIC	Tecnologia da Informação e Comunicação
IDAS	Intelligence Data Advanced Solution
IEEE	Institute of Electric and Electronic Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
JSON	JavaScript Object Notation
LWM2M	Lightweight Machine to Machine
M2M	Machine to Machine
MQTT	Message Queue Telemetry Transport
NGSI	Next Generation Service Interfaces
OCB	Orion Context Broker
POO	Programação Orientada à Objetos
QoS	Quality of Service
RAM	Random Acess Memory

REST	Representational State Transfer
RFID	Radio Frequency Identification
ROM	Read Only Memory
SOA	Service Oriented Architecture
STH	Short Time Historic
TCP/IP	Transmission Control Protocol/Internet Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VANETs	Vehicular Ad-hoc Networks

Sumário

1 Introdução

A Internet das Coisas (IoT) é um paradigma tecnológico imaginado como uma rede global de dispositivos que interagem entre si e com usuários, coletando informações do ambiente ou executando tarefas. Tal paradigma, adequa-se a aplicações de vários domínios, dentre eles, automação residencial e industrial, gerenciamento energético, cuidados médicos, etc. (ZANELLA et al., 2014).

Em virtude dos avanços da eletrônica de baixa potência e dos protocolos de rede, a quantidade de pesquisadores e indústrias interessadas nesse novo paradigma aumentou, o que consequentemente atraiu investimento em vários setores da economia, uma vez que sua principal característica é de conectar os mundos físico e digital por meio de dispositivos inteligentes, sejam eles sensores, atuadores ou uma combinação deles (HAN; CRESPI, 2017).

IoT permite aos indivíduos, sociedade e mundo dos negócios novas oportunidades para explorar um novo volume de dados, bem como desenvolver aplicações e novos serviços que constroem um ambiente de forma a tornar a sociedade mais inteligente (AHLGREN; HIDELL; NGAI, 2016). Além disso, seu objetivo é o aumento da confiabilidade, desempenho e segurança de cidades inteligentes e sua infraestrutura (BOTTA et al., 2016).

Em virtude do crescimento da densidade populacional nas cidades urbanas, torna-se necessário o incremento nos recursos de infraestrutura da cidade de forma a suprir as necessidades dos cidadãos, isto, implica no aumento de dispositivos digitais utilizados e.g. sensores, atuadores. Estes, devido ao seu desenvolvimento tecnológico aprimoram as cidades inteligentes (ARASTEH et al., 2016).

Apesar de existirem algumas plataformas para auxiliar o desenvolvimento de aplicações inteligentes, ainda é uma tarefa árdua sua configuração e até mesmo o seu desenvolvimento em si. Para esse propósito, existem plataformas que se propõem a fornecer meios que facilitem o desenvolvimento das aplicações, abstraindo detalhes para que o desenvolvedor tenha somente

como objetivo o desenvolvimento em si. A plataforma FIWARE[1], que será abordada nesse trabalho, é uma proposta para acelerar o desenvolvimento de aplicações em domínios como indústria e cidades inteligentes. Esta fornece componentes para facilitar o desenvolvimento de soluções que envolvem objetos inteligentes. Algumas de suas facilidades estão relacionadas a análise de grande volume de dados e fornecimento de medidas em tempo real, manipulação de informações de contexto, análise em tempo real, informações vindas dos sensores, ações dos atuadores e controle de acesso. Além disso, suas APIs e componentes tornam mais fácil a comunicação com dispositivos inteligentes, bem como a manipulação de dados do contexto.

Em (LEITE, 2019) e (SILVA, 2019) são relatadas duas aplicações desenvolvidas usando FIWARE. Estas, respectivamente, tratam de iluminação de uma praça com lâmpadas inteligentes, que aumentam sua intensidade para compensar uma vizinha que esteja com avaria, e uma sala de aula com ar-condicionado inteligente que ajusta seu uso levando em consideração o tempo da aula e a quantidade de pessoas dentro da mesma. Ambas as aplicações foram desenvolvidas utilizando a linguagem de programação Java. Uma vez que o acesso à API REST do FIWARE envolve o tratamento de objetos JSON, bem como o uso de requisições HTTP, seu código torna-se complexo e pouco modular sendo poluído por serialização/deserialização de objetos JSON, mesmo fazendo uso de bibliotecas dedicadas a lidar com objetos no formato JSON e requisições HTTP. Isto nos motivou a propor um *framework* orientado a objetos o qual é descrito neste trabalho.

1.1 Objetivos

Este trabalho tem como objetivo o desenvolvimento de um solução que busque facilitar o desenvolvimento de aplicações inteligentes utilizando a plataforma FIWARE, porém, abstraindo detalhes de seus componentes ao desenvolvedor. De uma forma mais simplificada, esse *framework* irá tratar as entidades da plataforma FIWARE como objetos da programação orientada a objetos (POO).

1.1.1 Objetivos específicos

- Realizar um estudo sobre IoT e cidades inteligentes, assim como também sobre a plataforma FIWARE e seus componentes relacionados com IoT.
- Definir, modelar e implementar o protótipo de um *framework* orientado a objetos que simplifique a programação de sistemas IoT para cidades inteligentes e que use componentes FIWARE como *back end*.
- Como prova de conceito, implementar as aplicações relatadas em (LEITE, 2019) e (SILVA, 2019) usando o *framework* proposto, assim como fazer uma comparação com as mesmas,

[1] https://www.fiware.org/developers/

de forma que nos permita avaliar os ganhos em termos de facilidade de desenvolvimento e modularidade dos sistemas.

1.2 Metodologia

Para o desenvolvimento deste trabalho, inicialmente foi realizado um estudo sobre IoT, FIWARE e algumas ferramentas que seriam necessárias para sua configuração e execução, como o software Docker[2]. Para a compreensão da FIWARE, seus componentes foram analisados em termos teóricos e práticos, por meio dos tutoriais[3] que o seu site disponibiliza. Como forma de coletar os requisitos do *framework*, foram realizados os estudos das aplicações definidas em (LEITE, 2019) e (SILVA, 2019), dessa forma, foram determinados os aspectos que contribuiriam para obtenção de um código mais claro e limpo. Foi definido que o *framework* teria Java como linguagem de programação e então foram definidas as suas principais funcionalidades e respectivas prioridades. Tais funcionalidades obedecem a documentação do NGSI[4]. Por fim, um protótipo do *framework* foi desenvolvido e então validado e avaliado com reimplementações dos sistemas relatados em (LEITE, 2019) e (SILVA, 2019).

1.3 Estrutura do Documento

O restante deste documento está estruturado em capítulos e seções. No capítulo 2, é realizado um estudo sobre IoT, abordando aspectos como arquitetura e protocolos. No mesmo capítulo, é realizado um estudo sobre cidades inteligentes, abordando os aspectos que a constituem, seus desafios e domínios de aplicações. No capítulo 3, é realizado um estudo sobre algumas das principais componentes da plataforma FIWARE. No capítulo 4, são apresentados como estudo de caso, duas aplicações envolvendo IoT e a plataforma FIWARE. No capítulo 5, é apresentada a proposta do *framework*, bem como os detalhes de sua implementação e a reimplementação dos sistemas do capítulo 4. No capítulo 6, são realizadas as considerações finais e contribuição em trabalhos futuros.

[2] Disponível em: www.docker.com/
[3] Disponível em: https://fiware-tutorials.readthedocs.io/en/latest/
[4] Protocolo padrão do FIWARE, disponível em: http://telefonicaid.github.io/fiware-orion/api/v2/stable/

2

IoT e Cidades Inteligentes

No presente capítulo deste trabalho, serão abordados os aspectos relacionados a IoT e Cidades Inteligentes. Estes foram divididos, respectivamente em duas seções. Na seção de IoT são abordados aspectos como sua definição, arquitetura, protocolos e seus desafios. Na seção de Cidades Inteligentes é abordada sua definição, aspectos, desafios e por fim, os domínios de aplicações.

2.1 IoT

O termo Internet das Coisas (do Inglês, *Internet of Things* - IoT) surgiu pela primeira vez em uma apresentação realizada por Kevin Ashton em 1999 (ASHTON et al., 2009). No entanto, mesmo após anos, a literatura disponível ainda não possui definição concreta, segundo (ATZORI; IERA; MORABITO, 2017) a definição que melhor se adequa atualmente é: "Uma estrutura conceitual que aproveita a disponibilidade de dispositivos heterogêneos e soluções de interconexão, bem como objetos físicos fornecendo uma base de informações compartilhada em escala global para suportar o design de aplicações envolvendo no mesmo nível virtual pessoas e representações de objetos."[1]

Segundo (HOLLER et al., 2014) IoT é um termo usado para denominar a conexão de dispositivos heterogêneos abrangendo vários aspectos relacionados à tecnologia. Num futuro próximo, imagina-se que IoT será uma rede de dispositivos que se comunicam com objetos do mundo real, trocando informações, interagindo com pessoas e dando suporte para negócios. Entretanto, deve ficar claro que IoT não se trata de uma nova internet, e sim de um novo paradigma inserido nela de tal forma que se torna uma extensão da mesma. Esta, também está relacionada com as tecnologias nas quais serão aplicadas e como será realizado seu monitoramento remoto e controle com o foco em premissas de inovação abertas, onde parte desse foco dedica-se ao

[1] tradução livre

processamento em ambientes fechados, tal como automação industrial, convergindo assim a um conceito de "Intranet das Coisas".

Levando em consideração os tipos de aplicações e serviços oferecidos, percebe-se que as possibilidades para a IoT são infinitas, limitando-se até onde a imaginação permite. No paradigma M2M (*Machine to Machine*) podemos enxergar domínios de aplicação que surgem da necessidade de indústrias, pessoas e a sociedade, atendendo tanto os interesses locais quanto globais. Tais aplicações podem direcionar seu foco para segurança, otimização de processos, redução de custos e etc.Algumas tendências de aplicação são a agricultura urbana, robótica e segurança alimentar (HOLLER et al., 2014).

De um ponto de vista conceitual, IoT se ergue sobre os pilares relacionados às características comuns de objetos inteligentes: a possibilidade de ser identificado por outros dispositivos, conseguir se comunicar com os demais dispositivos e interagir, tanto entre eles mesmos, quanto com usuários que de alguma forma têm acesso a esses recursos (MIORANDI et al., 2012).

Segundo (MIORANDI et al., 2012) um objeto inteligente é tipificado por:

- Possuir características físicas (tamanho, forma, etc).
- Dispor de mecanismo que permite a descoberta de outros dispositivos, bem como a capacidade de ser descoberto e de se comunicar, respondendo ou enviando mensagens.
- Possuir uma identificação única.
- Estar associado a pelo menos um nome e um endereço. O nome, é uma identificação que pode ser lida por humanos e usada para propósitos de raciocínio e lógica. O endereço é uma identificaçao a ser lida pela máquina, usado para comunicação dos objetos inteligentes.
- Possuir capacidades computacionais, que vão desde a capacidade de associar uma determinada mensagem a um remetente (como nas tarjas RFID's passivas) a tarefas de cômputo mais exigentes, como a descoberta de serviços e gerenciamento de recursos em rede.
- Capacidade de sentir fenômenos físicos do ambiente que foi instalado (no caso de sensores) ou disparar ações a partir das condições do ambiente (no caso de atuadores).

2.1.1 Arquitetura orientada a serviços

Tendo em vista que o objetivo do IoT é conectar dispositivos heterogêneos em rede, a arquitetura orientada a serviços (do Inglês, *Service Oriented Architecture* - SOA) dá o suporte necessário que pode ser aplicado na integração entre dispositivos e sistemas (XU; HE; LI, 2014). Contudo, à medida que o serviço IoT se torna maior e mais complexo, uma maior quantidade de dispositivos heterogêneos são envolvidos, requerendo assim uma forte capacidade de integração e adaptabilidade a um ambiente altamente dinâmico (CHENG et al., 2016).

Como a natureza da IoT é descentralizada, heterogênea e voltada a eventos, a SOA se adequa perfeitamente, pois permite alcançar interoperabilidade entre dispositivos diversos (XU; HE; LI, 2014). Uma outra abordagem vista na literatura, é a utilização de *web services* para serviços IoT (ZANELLA et al., 2014). Nesta, os *web services* permitem a criação de um sistema flexível e interoperável que pode ser estendido aos nós de IoT por meio de um paradigma baseado em web conhecido como *Representational State Transfer* (ReST) (FIELDING; TAYLOR, 2000). Este, permite publicação, descoberta, seleção e composição dos serviços oferecidos pelos dispositivos IoT (CHEN; GUO; BAO, 2016).

Em geral, SOA é um padrão arquitetural usado para integração de sistemas e automação de processos que permite, de uma maneira mais fácil, a composição e coordenação de serviços sobre uma rede (CHENG et al., 2016). Em sistemas IoT baseados em SOA, cada dispositivo pode desempenhar tanto o papel de consumidor, quanto o papel de produtor, e até mesmo ambos os papéis são permitidos, oferecendo serviços, compartilhando recursos e interagindo com consumidores por meio de API's compatíveis tais como ReST e CoRE (CHEN; GUO; BAO, 2016).

No design de uma API para *web services* é comum o uso do estilo arquitetural ReST. Possuir uma API que respeita os princípios do ReST, significa ter uma API ReST, ou uma API "RESTful"(MASSE, 2011). Em um design RESTful, cada objeto é representado como um único recurso que pode ser consultado de uma maneira uniforme por meio dos métodos HTTP (GET, POST, PUT, DELETE, etc.) e de URI's únicas para acessá-los. As ações para o acesso aos objetos de uma API RESTful retornam, caso seja necessário, o objeto estruturado, ou o resultado da ação sobre o mesmo. Tal objeto estruturado pode ser representado no formato JSON. (ONG et al., 2015).

O formato JSON (do inglês - *Javascript Object Notation* (json.org, 2001), é um formato leve para troca de dados. De fácil leitura e escrita para humanos e de fácil geração e *parsing* para máquinas. É baseado na terceira edição do padrão ECMA-262 da linguagem de programação *Javascript*. JSON é um formato que é totalmente independente da linguagem, no entanto, faz uso de convenções familiares aos programadores de linguagens da família da linguagem C.

Código 1 – Código java necessário para uma requisição à uma API ReST utilizando *payload* no formato JSON.

```
Client client = ClientBuilder.newClient();
Entity payload = Entity.json(
"{
    \"type\": \"Room\",
    \"id\": \"Bcn-Welt\",
    \"temperature\": {    \"value\": 21.7  },
    \"humidity\": {    \"value\": 60  },
    \"location\": {    \"value\": \"41.3763726, 2.1864475\",
    \"type\": \"geo:point\",
    \"metadata\":
    {      \"crs\":
    {        \"value\": \"WGS84\"
    }
    }}
}"
    );
Response response = client.target(
"http://orion.lab.fiware.org/v2/entities?options=")
.request(MediaType.APPLICATION_JSON_TYPE)
.post(payload);
```

Fonte: (FIWARE FOUNDATION, 2018)

No código 1, uma entidade de `id`: Bcn-Welt e `type`: Room e que possui os atributos `temperature, humidity` e `location` é submetido via requisição POST. Tal requisição, criará essa entidade no *broker* especificado.

2.1.2 Protocolos IoT

Devido à grande quantidade de objetos inteligentes, a integração dos mesmos apresenta um desafio, uma vez que sua troca de dados ocorre por diversos meios e protocolos de comunicação. Assim sendo, o IEEE e o IETF estão trabalhando no desenvolvimento de protocolos padronizados (RAHMAN; OZCELEBI; LUKKIEN, 2016). Segundo (SKARMETA; HERNANDEZ-RAMOS; MORENO, 2014), as seguintes características comuns em todos os novos protocolos IoT são:

- **Comunicação de baixo consumo**: A maioria dos dispositivos fazem uso de bateria, mas é desejável que o consumo energético seja muito baixo para evitar a drenagem total da baterias e consequentemente a interrupção do funcionamento. Baterias maiores e com maior capacidade de armazenamento não são leves e demandam maior espaço físico, e assim sendo, protocolos IoT que permitem baixo consumo energético aos dispositivos sem fio se mostra uma tarefa desafiadora.

- **Comunicação altamente confiável e rápida**: Para transmissão IoT, a conectividade à internet deve ser rápida e confiável. Porém, a principal fonte de baixa confiabilidade é da interferência externa devido à falha no sinal, o que pode encerrar a comunicação, tornando assim necessária a retransmissão dos dados, o que acarretará em um consumo energético maior.

- **Comunicação habilitada para a Internet**: É de grande importância que a comunicação máquina a máquina por meio de conexões pela internet adote uma abordagem universal, uma vez que é necessário que a comunicação entre os diversos objetos deve ser facilitada (PALATTELLA et al., 2013).

Segundo (NAIK, 2017), os protocolos MQTT, CoAP, AMQP e HTTP são largamente aceitos, estes, se encontram no topo da pilha de protocolos da figura 1 e serão brevemente descritos juntamente com o protocolo 6LoWPAN.

Figura 1 – Pilha de protocolos para sistemas IoT

Fonte: (NAIK, 2017)

- **MQTT** (*Message Queue Telemetry Transport*): É um protocolo de transmissão de mensagens leve, aberto, simples e projetado para que sua implementação seja realizada de uma maneira fácil. Este, roda sobre TCP/IP ou sobre qualquer outro protocolo de rede que forneça conexão bidirecional, sem perda e ordenação. O protocolo MQTT faz uso do padrão *publish-subscribe*, o que fornece transmissão de mensagem de um para muitos. Além disso, o protocolo MQTT minimiza a sobrecarga e também notifica as partes interessadas quando um evento de seu interesse ocorre (KRAIJAK; TUWANUT, 2015).

- **CoAP** (*Constraint Application Protocol*): É um protocolo web para ser usado com nós e redes limitados (baixo consumo energético, perda de pacotes). Normalmente, os nós possuem controladores de 8 bits com pequenas quantidades de memória RAM (*Random Access Memory*) e memória ROM (*Read Only Memory*), enquanto a rede possui alta perda de pacotes e a vazão típica é de 10kbps (PALATTELLA et al., 2013). Esse protocolo é projetado para aplicações Máquina à Máquina (M2M) comuns no âmbito de Cidades

inteligentes e automação de prédios. O protocolo CoAP é projetado para desempenhar o papel de uma interface com o protocolo HTTP para integração com a web, satisfazendo requisitos como suporte a multicast, baixa sobrecarga e simplicidade para ambientes restritos (KRAIJAK; TUWANUT, 2015).

- **AMQP** (*Advanced Message Queuing Protocol*): É um protocolo corporativo projetado para oferecer confiabilidade, segurança, provisionamento e interoperabilidade (FOSTER, 2015). Além disso, AMQP suporta tanto a arquitetura *Request/Response* quanto a arquitetura *Publish/Subscribe* e oferece uma vasta quantidade de características relacionadas a mensagens, tais como enfileiramento confiável, mensagens *Publish/Subscribe* baseadas em tópicos, transações e roteamento flexível (FOSTER, 2015). O sistema de comunicação AMQP requer que ou o produtor, ou o consumidor crie uma *exchange* com um dado nome, e então realize transmissões para esse nome, assim, produtores e consumidores fazem uso dessa *exchange* para descobrirem um ao outro. Assim que o consumidor cria uma fila, esta é anexada na *exchange* ao mesmo tempo, então assim que uma mensagem é recebida pela *exchange*, uma correspondente deve chegar na fila por um processo chamado de *binding*. As trocas de mensagem no AMQP podem ser realizadas de várias formas diferentes: diretamente, em modo *fanout*, por tópico ou baseado em cabeçalhos (NAIK, 2017). O protocolo AMQP requer um cabeçalho fixo de 8 bytes com um pequeno *payload* para as mensagens, indo até o tamanho máximo do servidor/*broker* escolhido (LUZURIAGA et al., 2015), (MARSH et al., 2008). A comunicação entre o cliente e o *broker* é orientado a conexão, onde a confiabilidade é uma das principais características e oferece dois níveis de Qualidade de Serviço (QoS) para a entrega das mensagens: os formatos *Unsettle* (Não confiável) e *Settle* (Confiável) (FOSTER, 2015).

- **HTTP**: É um dos mais importantes protocolos da Web (HAN, 2015). Este suporta a arquitetura Requisição/Resposta RESTful. Similar ao CoAP, HTTP usa Identificadores de Recursos Universais (URI) ao invés de tópicos. O servidor envia dados por meio de uma URI e o cliente o recebe por uma URI particular. HTTP é um protocolo baseado em texto e não define tamanho do cabeçalho e o *payload* das mensagens é definido pelo servidor escolhido ou tecnologia de programação (NAIK, 2017).

- **6LoWPAN**: É um protocolo de rede que permite conexão direta à internet usando padrões abertos e que define mecanismos de encapsulamento e compressão de cabeçalhos. Além disso, possui flexibilidade para operar sobre múltiplas plataformas de comunicação, incluindo Ethernet, WiFi e 802.15.4 (PALATTELLA et al., 2013). Devido à necessidade de dispositivos de baixo consumo energético, este protocolo substituiu o IPv6, pois dá suporte a endereços com diferentes comprimentos, baixa largura de banda, topologias em estrela e malha, dispositivos que demandam o uso de bateria, baixo custo, posições de nós desconhecidos, conexões com baixa confiabilidade e longos períodos ociosos quando a comunicação é desligada para economia de energia (PALATTELLA et al., 2013).

2.1.3 Desafios da IoT

De acordo com (KHAN et al., 2012), a IoT possui o potencial de modificar a forma da internet e trazer enorme benefício econômico. No entanto, a IoT enfrentará diversos desafios, tais como:

- **Gerenciamento de nomes e identidades**: IoT conectará bilhões de dispositivos que fornecem os mais diversos serviços. Estes, por sua vez, devem possuir identidade única na internet, e portanto, um sistema eficiente para atribuição e gerenciamento de identidades únicas a um grande número de dispositivos é necessário.

- **Interoperabilidade e Padronização**: Muitos fabricantes fornecem dispositivos que fazem uso de tecnologias proprietárias, dessa forma, impossibilitando ser utilizado em conjunto com demais tecnologias. Assim, a padronização da IoT é importante no aspecto de permitir melhor interoperabilidade entre todos os dispositivos.

- **Privacidade das informações**: A IoT utiliza diversas tecnologias para identificação de objetos e.g., RFID, código de barras, etc. Como os objetos do cotidiano irão carregar tarjas de identificação, bem como informações específicas e próprias dos objetos, é necessário adotar uma política de privacidade referente ao recebimento de informações e acesso indevido às mesmas.

- **Objetos de segurança e segurança**: A IoT consiste de uma vasta área geográfica coberta por dispositivos espalhados na mesma, assim, é necessário prevenir acesso de intrusos que possam causar dano físico aos objetos ou mudar a forma de operação destes.

- **Confidencialidade e criptografia de dados**: Os dispositivos sensores, são responsáveis por capturar medições do ambiente e transferir para uma unidade de processamento por meio de um sistema de transmissão. Este, entretanto, deve fornecer um mecanismo de criptografia apropriado que garanta a integridade dos dados. O serviço IoT deve determinar quem possui autorização para visualizar os dados, consequentemente, protegendo-o de acesso externo indevido.

- **Segurança de rede**: Os dados oriundos dos dispositivos são enviados por meio de redes com ou sem fio. O sistema de transmissão deve ser capaz de tratar os dados de uma grande quantidade de dispositivos sem que haja perda de dados devido ao congestionamento. Além disso, deve garantir medidas adequadas de segurança que previna interferência externa e monitoramento.

- **Espectro**: Os dispositivos irão necessitar de um espectro dedicado para a transmissão de dados sobre o meio sem fio. Devido à disponibilidade limitada do espectro, um mecanismo de alocação dinâmica de espectro que permita bilhões de dispositivos comunicarem-se sobre o meio sem fio é necessária.

- **Sustentabilidade da IoT**: Devido ao aumento da quantidade de dispositivos e serviços na internet, o consumo de energia está aumentando à uma alta taxa. Assim, a IoT futuro causará, um aumento significante no consumo energético de rede, trazendo a necessidade da adoção de tecnologias sustentáveis para que a eficiência energética aumente.

2.2 Cidades Inteligentes

Devido à sua habilidade de monitoramento, gerenciamento, controle de dispositivos e ações disparáveis por meio da captura de *streams* de dados massivas, novas aplicações de IoT estão permitindo uma maior quantidade de iniciativas relacionadas às cidades inteligentes ao redor do mundo (KIM; RAMOS; MOHAMMED, 2017).

O termo cidades inteligentes tem sido frequentemente usado tanto academicamente, quanto na indústria. No entanto, ainda não há um claro entendimento do termo (CHOURABI et al., 2012) pois este conceito ainda está emergindo e o trabalho de definir e conceitualizar ainda está em progresso (BOULTON; BRUNN; DEVRIENDT, 2011), (HOLLANDS, 2008). Segundo (HARRISON et al., 2010) uma cidade inteligente é: *"A city connecting the physical infrastructure, the IT infrastructure, the social infrastructure, and the business infrastructure to leverage the collective intelligence of the city"*. Segundo (ZANELLA et al., 2014), o objetivo final de uma cidade inteligente é fazer um melhor uso dos recursos públicos, melhorando, assim a qualidade dos serviços oferecidos aos cidadãos e ao mesmo tempo, reduzindo custos operacionais de administrações públicas. Esse objetivo, é alcançado por meio de uma implantação de um IoT urbano, ou seja, de uma infraestrutura de Tecnologia da Informação e Comunicação (TIC).

Junto com a grande tendência de urbanização, pode-se notar também uma revolução digital, impulsionada pelo desenvolvimento das TICs, e pela disseminação de dispositivos fixos e móveis conectados, a chamada hiperconectividade, tanto entre pessoas como entre máquinas (M2M), transformando o modo como o sistema produtivo e a sociedade se articulam. Tal impacto da revolução digital sobre um mundo em acelerada urbanização sustenta a emergência do conceito cidade inteligente (CUNHA et al., 2016).

2.2.1 Aspectos de uma Cidade Inteligente

Em (GIFFINGER; PICHLER-MILANOVIĆ, 2007), são definidos seis aspectos que tornam uma cidade inteligente. Estes, determinam o comportamento da cidade e crescimento como um todo. Além disso, determinam as tendências e oportunidades do ponto de vista de inovação e desenvolvimento (PELLICER et al., 2013). Os aspectos que tornam uma cidade inteligente serão explicados abaixo:

- **Economia Inteligente**: Este, tem como foco o espírito inovador, imagem e marcas comerciais, produtividade, flexibilidade, imersão e capacidade de transformação.

- **Governança Inteligente**: Inclui todos os procedimentos relacionados à participação pública como: tomada de decisão, serviços públicos e sociais, transparência governamental, perspectivas e estratégias políticas.

- **Pessoas Inteligentes**: Cidades sustentáveis precisam de capital humano qualificado apropriadamente para que a sua evolução e crescimento possa ser acompanhada. Portanto, os seguintes aspectos são levados em consideração: nível de qualificação, afinidade com aprendizagem ao longo da vida, pluralidade étnica e social, flexibilidade, criatividade, cosmopolitismo e participação na vida pública.

- **Mobilidade Inteligente**: Na sociedade atual, a mobilidade é um serviço essencial, porém, não podemos afirmar que este constitui um sistema sustentável (alterações climáticas, emissões, ruídos, congestionamentos e acidentes). Estas, são motivações que levam à pesquisa de novas tecnologias de transporte e sistemas de gerenciamento de tráfego urbano. Algumas tendências nas cidades inteligentes são: acessibilidade local e internacional, disponibilidade de infraestrutura e sistemas de transporte sustentáveis, seguros e inovadores.

- **Ambientes Inteligentes**: Cidades inteligentes possuem uma ampla gama no desenvolvimento e evolução no campo de eficiência energética e gerenciamento de recursos. Entre essas linhas de desenvolvimento, podemos citar: atratividade das condições naturais, poluição, proteção ambiental e gestão sustentável de recursos.

- **Vida Inteligente**: No desenvolvimento de cidades inteligentes, os cidadãos são o ponto chave. As aplicações de cidades inteligentes devem melhorar a qualidade de vida dos mesmos por meio de instalações culturais, condições de saúde, segurança individual, qualidade da habitação, instalações educativas, atratividade turística e coesão social.

2.2.2 Desafios das Cidades Inteligentes

Dados urbanos de diferentes domínios são coletados como ponto de partida para cidades inteligentes. De um ponto de vista geral, por meio da integração, análise e vitalização dos dados, pesquisadores podem projetar de forma mais eficiente e inteligente aplicações ou sistemas para cidades inteligentes (YIN et al., 2015). Além dos desafios científicos e tecnológicos, o incentivo à colaboração da população e o bom uso de recursos públicos surgem como desafios sociais e culturais (KON; SANTANA, 2016). Segundo (KON; SANTANA, 2016), os desafios das cidades inteligentes são:

- **Privacidade**: Como resultado da crescente confiança das instituições públicas e privadas na interação digital com cidadãos e consumidores, a pesquisa sobre privacidade têm avançado nas últimas décadas. Dentro desse campo, três dimensões influenciam as pessoas sobre sua privacidade: Tipos de dados envolvidos, propósito da coleta e uso dos dados e os órgãos ou pessoas que coletam e usam os dados (ZOONEN, 2016).

- **Segurança**: A cidade inteligente deve ser capaz de defender as informações contra acesso sem autorização, divulgação, modificação, inspeção e aniquilação (ZHANG et al., 2017). Os requisitos de segurança e privacidade, tais como, confidencialidade, integridade, não repúdio, disponibilidade, controle de acesso e privacidade, devem ser satisfeitos (MARTÍNEZ-BALLESTÉ; PÉREZ-MARTÍNEZ; SOLANAS, 2013).

- **Gerenciamento dos dados**: As cidades inteligentes tem como uma das principais características o grande volume de dados. Estes, podem vir dos cidadãos, sensores e de fluxos de imagens, sendo atribuído a estes rótulos estruturados, semi-estruturados e não-estruturados, respectivamente. Os desafios dessa área residem no armazenamento, processamento, modelos e confiabilidade dos mesmos (KON; SANTANA, 2016).

- **Escalabilidade**: Dentro das próximas décadas, o número de dispositivos conectados continuará aumentando, exigindo assim, um alto nível de escalabilidade (BALAKRISHNA, 2012). Além disso, com o crescimento da população o número de usuários, serviços e dados armazenados também irão crescer (SANTANA et al., 2018).

- **Heterogeneidade**: Cidades inteligentes são caracterizadas pela heterogeneidade de redes, aplicações, dispositivos, plataformas etc. Pode-se tomar como exemplo as redes de veículos que podem exigir redes ad-hoc, como VANETs, enquanto dispositivos sem fio de curta distância podem funcionar no Zigbee. As aplicações legadas e sua integração com tecnologias emergentes também serão um desafio significativo (BAWANY; SHAMSI, 2015).

- **Manutenção e Implantação da Infraestrutura**: A criação de uma cidade inteligente trará a necessidade de diversos investimentos para a implantação da infraestrutura necessária. A implantação de sensores, atuadores e a melhoria das redes, bem como a integração e a coleta de dados da infraestrutura existente. Após a implantação, também será necessário fazer a manutenção dos componentes da infraestrutura, uma vez que estão sujeitos a falhas parciais e avarias (KON; SANTANA, 2016).

- **Custos**: Segundo (BAWANY; SHAMSI, 2015), cidades inteligentes envolvem a aquisição de uma grande infraestrutura de TI. Enorme investimento financeiro deve ser feito para o funcionamento do sistema. Milhões de sensores, milhares de equipamentos de rede e dispositivos de computação serão necessários para uma conectividade ponta a ponta. Além disso, a necessidade de profissionais de TI e consultorias representa despesas consideráveis, bem como o custo operacional e de manutenção de um sistema tão grande será muito maior. Para que os requisitos de confiabilidade e eficiência sejam satisfeitos, mais recursos serão necessários, gerando assim maiores despesas gerais.

- **Colaboração**: Os cidadãos tem um importante papel na transformação de uma cidade comum em uma cidade inteligente. Estes, podem atuar como fontes de informações, sejam

sobre eles próprios ou sobre o mundo físico, por exemplo, por meio de seus dispositivos móveis. A modelagem, entendimento e influência no comportamento humano, bem como o fornecimento de informações e feedback, cultiva, assim, uma cultura de confiança em tecnologias de cidade inteligente que são desafios-chave que se baseiam no trabalho da psicologia, experiência do usuário e computação social (NAPHADE et al., 2011).

2.3 Domínios de Aplicações IoT e cidades inteligentes

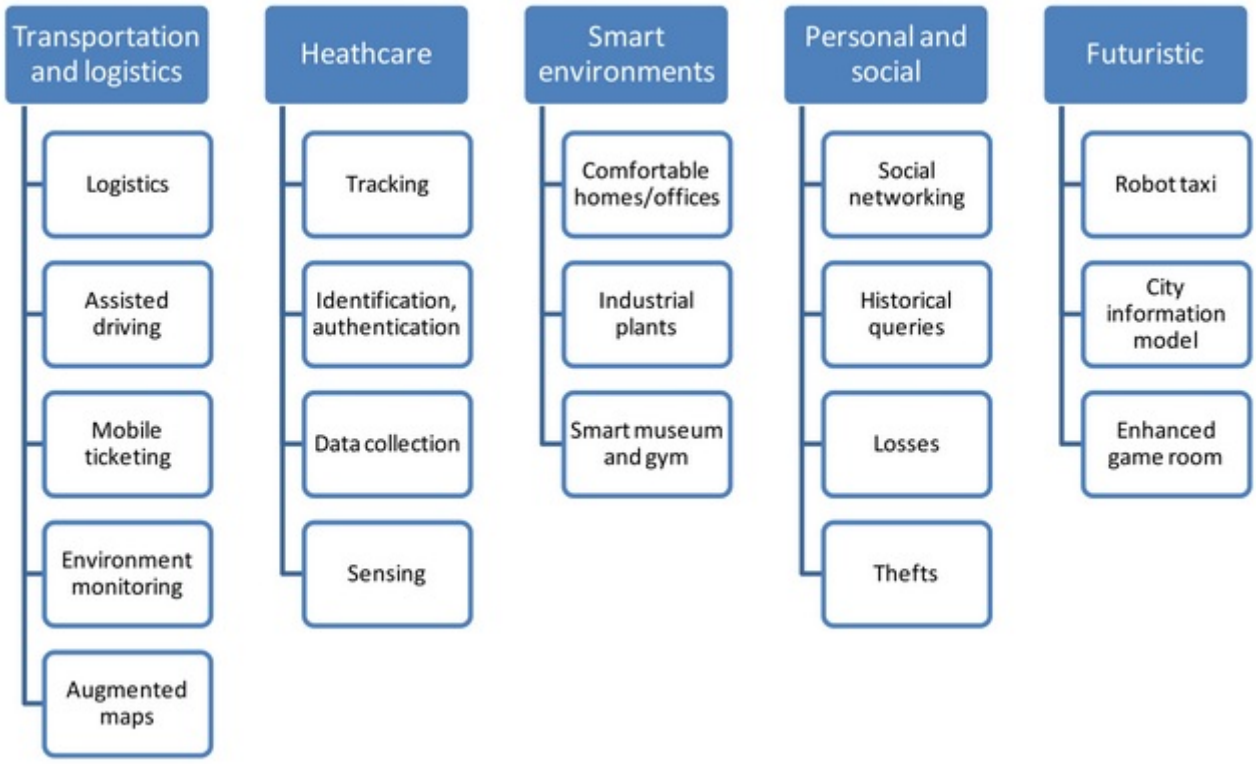
Neste momento, apenas a nossa imaginação atua como fator limitante dos domínios de aplicações IoT e Cidades Inteligentes (WHITMORE; AGARWAL; XU, 2015). Segundo (ATZORI; IERA; MORABITO, 2010) o potencial oferecido pelas aplicações IoT é enorme, estas aplicações, tem como um dos objetivos melhorar a qualidade de vida da sociedade. Ao oferecer a dispositivos a capacidade de se comunicarem entre si e elaborarem informações recebidas do ambiente, possibilita uma gama de aplicações que podem ser implantadas, estas, são dividas nos seguintes domínios:

- **Domínio de Transportes e logística**: Carros, trens e ônibus avançados, estão sendo equipados com sensores, atuadores e poder de processamento. Informações de tráfego coletadas das estradas por meio de tarjas e sensores são utilizadas para melhorias no trânsito, fornecimento de informações adequadas a turistas e monitoramento do status de mercadorias transportadas.
- **Domínio de Cuidados de Saúde**: Os benefícios obtidos pelas tecnologias IoT podem ser aplicados em rastreamento de objetos e pessoas (equipes e pacientes), identificação e autenticação de pessoas, detecção e coleta automática de dados (VILAMOVSKA et al., 2009).
- **Domínio de ambiente inteligente**: Ambiente inteligente é aquele que faz seu "emprego"fácil e confortável, utilizando-se de objetos inteligentes contidos seja em um escritório, uma casa, um ambiente industrial ou um ambiente de lazer.
- **Domínio Pessoal e Social**: Aplicações desse domínio são aquelas que permitem ao usuário manter e construir relacionamentos sociais. Esta, deve disparar o envio de mensagens para amigos, e que estes, saibam de eventos como mudança de casa, de emprego, ou até mesmo uma partida de futebol (WELBOURNE et al., 2009).

Entre estes domínios, existem aqueles que estão mais próximos de serem diretamente aplicáveis aos nossos hábitos atuais, e existe o domínio futurístico, que são de tecnologias ainda não prontas para serem implantadas, e portanto, podemos apenas imaginar como seria a sua implantação (ATZORI; IERA; MORABITO, 2010). Na figura 2, podem ser observados os domínios e

aplicações IoT, ao leitor que se interessar, em (ATZORI; IERA; MORABITO, 2010), é dada uma visão ainda mais completa do que foi citado neste trabalho.

Figura 2 – Domínios e Aplicações IoT



Fonte: (ATZORI; IERA; MORABITO, 2010)

O *framework* proposto neste trabalho, desenvolvido para a plataforma FIWARE, oferece uma infraestrutura para suporte ao desenvolvimento de aplicações no campo de IoT e cidades inteligentes. A plataforma FIWARE, bem como alguns de seus principais componentes serão detalhados no capítulo 3.

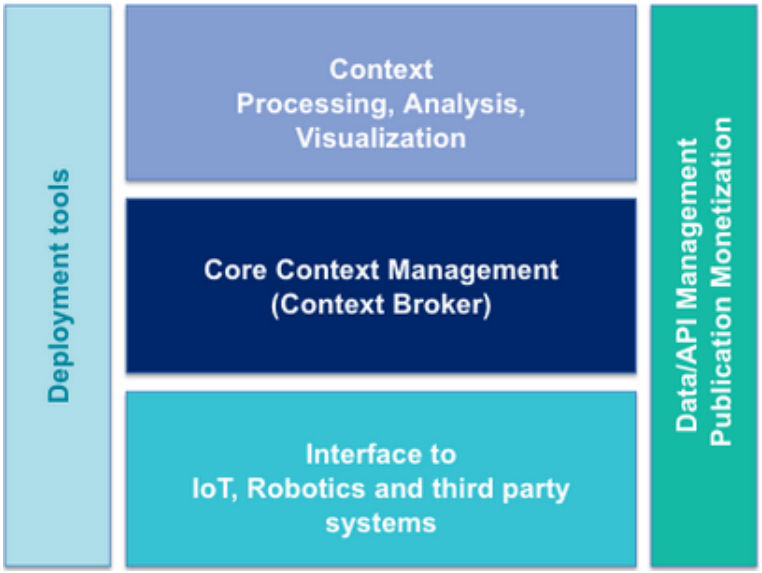
3
FIWARE

O FIWARE é uma iniciativa da Comissão Europeia (CE) cujo objetivo é facilitar o desenvolvimento de aplicações inteligentes por meio de uma infraestrutura aberta baseada em nuvem que oferece um catálogo de componentes chamados de Generic Enablers (GEs) (COLA et al., 2015). A adoção dos componentes da plataforma FIWARE, permite reduzir o tempo de desenvolvimento da solução como um todo, aumentando a modularidade, escalabilidade e a flexibilidade do sistema (FAZIO et al., 2015).

Segundo (FAZIO et al., 2015), foram investidos 80 milhões de euros pela CE, de modo a promover a adoção do Programa Acelerador (PA) do FIWARE. Este programa inclui 16 projetos em diferentes tópicos, tais como: Cidades Inteligentes, multimídia, assistência médica, agroalimentar etc. Além disso, encarrega-se de fornecer fundos para pequenas e médias empresas de TIC que se interessem em utilizar o FIWARE.

O FIWARE, é dividido em capítulos técnicos que oferecem um conjunto de habilitadores genéricos. Alguns deles, no entanto, estão sob encubadora e portanto, não cabem no escopo desse trabalho. Na figura 3, podem ser vistos de uma melhor forma os capítulos FIWARE.

Figura 3 – Capítulos FIWARE



Fonte: (FIWARE, 2016a)

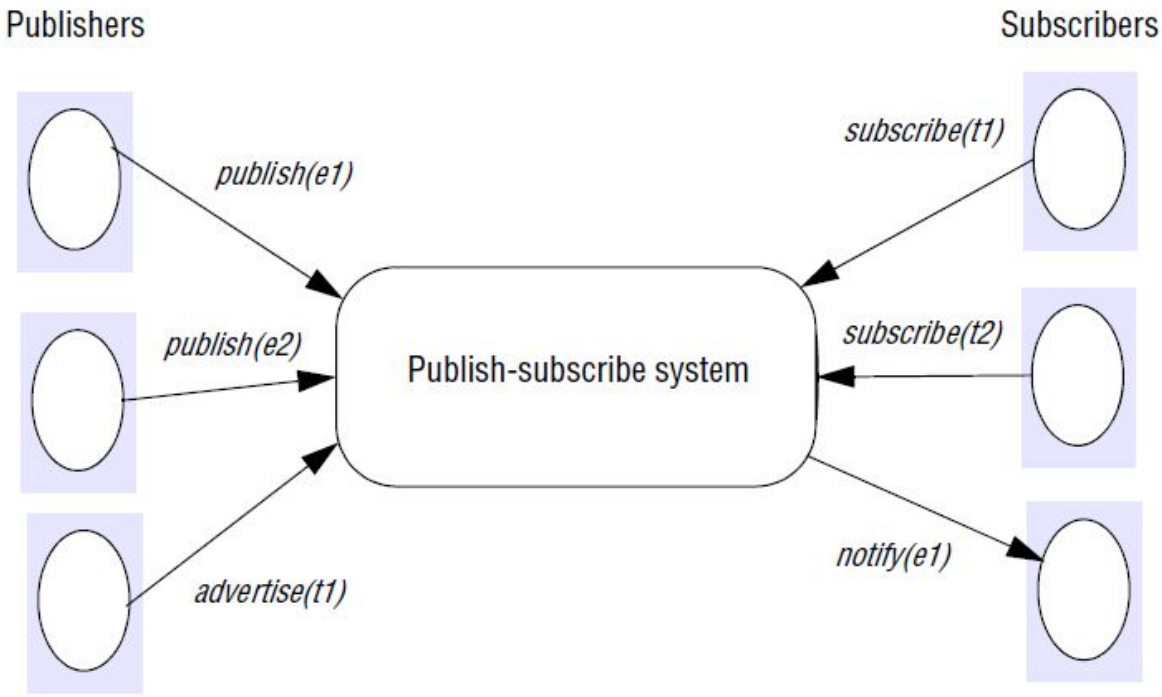
3.1 *Orion Context Broker*

Um *Context Broker* (CB) permite o gerenciamento de todas as informações de contexto em todo ciclo de vida, o que inclui registros, atualizações, subscrições e consultas. Por meio de sua utilização, é possível armazenar elementos de contexto e gerenciá-los com atualizações e consultas. Além disso, um CB é capaz de gerenciar subscrições a informações de contexto, assim, quando uma condição é disparada, ou seja, quando um valor é alterado, ou uma determinada quantidade de tempo passou, o *subscriber* recebe uma notificação (FERNÁNDEZ et al., 2016).

Nesse tipo de arquitetura, os publicadores (*publishers*) divulgam eventos estruturados para um serviço de evento e os assinantes (*subscribers*) demonstram interesse em eventos específicos por meio de assinaturas. A tarefa do sistema *Publish-Subscribe* é combinar as assinaturas com os eventos publicados e garantir a entrega correta das notificações de eventos. *Publish-Subscribe* lida bem com heterogeneidade, até mesmo com componentes na internet, pois basta que os objetos que gerem eventos publiquem informações sobre os eventos e os outros objetos se subscrevam a esses eventos e forneçam uma interface para receber e lidar com as notificações resultantes. Além disso, outra característica interessante do *Publish-Subscribe* é que as notificações sobre os eventos são enviados de forma assíncrona, para evitar que os *publishers* estejam sincronizados com os *subscribers*, então, necessariamente deve haver desacoplamento entre as partes, ou seja, tanto o *publisher*, quanto o *subscriber* precisam estar desacoplados (COULOURIS et al., 2011).

Na figura 4 é possível ter um melhor entendimento do *Publish-Subscribe*.

Figura 4 – Padrão Publish-Subscribe



Fonte: (COULOURIS et al., 2011)

Assim, o *Orion Context Broker (OCB)* é um sistema *Publish-Subscribe*. Ele atua como uma ponte que permite que aplicações externas gerenciem eventos IoT de uma forma simples, uma vez que esconde a complexidade das medições vindas dos recursos IoT (e.g. sensores) (FERNÁNDEZ et al., 2016).

Segundo (FERNÁNDEZ et al., 2016), o OCB possui duas interfaces API REST denominadas NGSI9 e NGSI10, que permitem as seguintes operações:

- **NGSI9**: Registro de conteúdo, descoberta, subscrição, atualização e cancelamento de subscrição da disponibilidade de contexto.
- **NGSI10**: Atualização, consulta, subscrição, atualização da subscrição e cancelamento da subscrição dos conteúdos.

3.1.1 O padrão NGSI

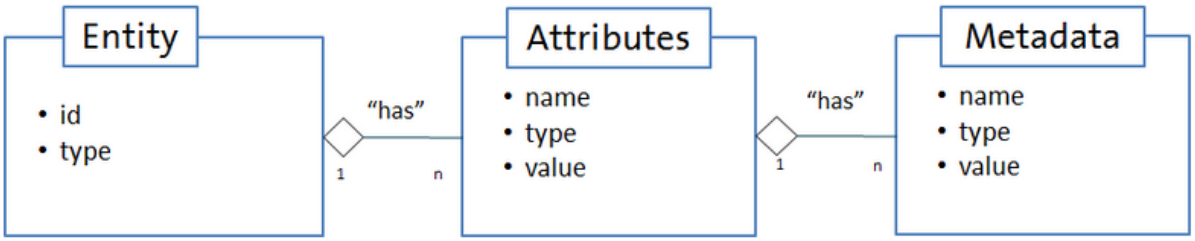
O FIWARE NGSI (do inglês - *The Next Generation Service Interfaces*) é responsável por todo o ciclo de vida das informações de contexto, isto inclui atualizações, consultas, registros e subscrições (FIWARE-NGSI, 2018). Sua API define:

- Um **modelo de dados** para informações de contexto, baseado em um modelo de informações simples usando a noção de entidades de contexto;

- Uma **interface de contexto de dados** para troca de informações por meio de operações de consulta, subscrição e atualizações;
- Uma **interface de disponibilidade de contexto** para troca de informação sobre informações de contexto.

Os principais elementos no modelo de dados NGSI são entidades, atributos e metadados. A relação entre os mesmos é mostrado na figura 5:

Figura 5 – Relação entre entidade, atributo e metadados do NGSI



Fonte: (FIWARE-NGSI, 2018)

Como pode ser visto na figura 5, a entidade é o principal elemento no modelo de informações do FIWARE NGSI. Esta, pode representar qualquer objeto físico ou lógico, como por exemplo, uma pessoa, um sensor, uma sala, etc. Cada entidade deve possuir um *id*. Também é permitido a entidade possuir um tipo, que são usados para descrever o tipo do objeto representado pela entidade. A combinação de id e tipo, identifica unicamente uma entidade.

Os atributos representam as propriedades de uma entidade. Estas, possuem nome, tipo, valor e metadados. O nome do atributo descreve a representação do atributo. O tipo do atributo representa o tipo do valor do atributo. E o valor do atributo contém o valor em si da propriedade, e também metadados, que descrevem propriedades do valor do atributo e.g. (precisão, data da última atualização, etc).

Os metadados são usados em várias partes do FIWARE NGSI, cada metadado deve possuir:

- **Nome**: Descreve o papel do metadado;
- **Tipo**: Descreve o tipo do metadado;
- **Valor**: Contém o metadado em si.

Uma entidade no padrão NGSI é representado por um objeto JSON como ilustrado no código 2:

Código 2 – Representação de uma entidade NGSI no formato JSON

```
{
"id": "entityID",
"type": "entityType",
"attr_1": <val_1>,
"attr_2": <val_2>,
...
"attr_N": <val_N>
}
```

O valor de um atributo NGSI pode ser representado da forma ilustrada no código 3:

Código 3 – Representação de um atributo NGSI no formato JSON

```
{
  "value": <...>,
  "type": <...>,
  "metadata": <...>
}
```

Operações de *Create*, *Read*, *Update* e *Delete* sobre entidades e atributos são permitidas. Estas são feitas via requisições HTTP à API REST do NGSI. Os códigos 4 e 5 ilustram uma requisição HTTP GET junto com sua resposta. Pode-se notar que é feita uma requisição para recuperar a entidade `Room1`, que possui os atributos `pressure` e `temperature`.

Código 4 – Acessando uma entidade persistida no *Orion* via curl

```
curl localhost:1026/v2/entities/Room1
-s -S -H 'Accept: application/json'
```

Código 5 – Resposta referente à requisição do código 4

```
{
    "id": "Room1",
    "pressure": {
        "metadata": {},
        "type": "Integer",
        "value": 720
    },
    "temperature": {
        "metadata": {},
        "type": "Float",
        "value": 23
    },
    "type": "Room"
}
```

Caso seja necessário atualizar uma entidade, uma requisição HTTP pode ser realizada informando, na URL, a entidade a ser atualizada, e, no corpo da requisição, os novos valores dos atributos como pode ser visto no código 6.

Código 6 – Atualização dos atributos da entidade `Room1`

```
curl localhost:1026/v2/entities/Room1/attrs
-s -S -H 'Content-Type: application/json' -X PATCH -d @- <<EOF
{
    "temperature": {
        "value": 26.5,
        "type": "Float"
    },
    "pressure": {
        "value": 763,
        "type": "Float"
    }
}
EOF
```

Assim, no código 6, é realizada uma atualização da entidade `Room1`, alterando os atributos `pressure` para o valor 763, e `temperature` para o valor 26,5.

Entre as operações disponibilizadas pela API NGSI, pode-se destacar as disponíveis na tabela 1:

Tabela 1 – Principais operações disponíveis na API NGSI

	Create	Read	Update	Delete
Entities	X	X	X	X
Attributes	X	X	X	X
Types	–	X	–	–
Subscriptions	X	X	X	X
Registrations	X	X	X	X

Na tabela 1, onde o X denota que a operação é permitida, caso contrário, a operação não é permitida. Note que são permitidas as operações de CRUD - (do inglês *Create, Read, Update and Delete*) em *Entities, Attributes, Subscriptions, Registrations*. Algumas destas, foram implementadas no *framework* proposto do capítulo 5.

3.2 Short Time Historic (STH)

O STH é um componente do FIWARE encarregado do gerenciamento (armazenamento e recuperação) de histórico de informações de dados de contexto, ou seja, valores de atributos de entidades registrados no OCB. Este, adiciona memória à plataforma FIWARE, possibilitando obter dados brutos e agregados sobre os valores que as entidades assumiram ao longo do tempo, tornando-o assim, o que pode ser chamado de um banco de dados de séries temporais (FIWARE, 2019c).

Toda a comunicação entre o STH e o OCB, assim como entre o STH e sistemas de terceiros usam as interfaces padronizadas NGSI9 e NGSI10. Apesar do STH suportar o armazenamento e recuperação de informações de contexto, sua principal capacidade e responsabilidade é a geração de séries temporais, bem como sua respectiva evolução no tempo sobre as informações de contexto dos atributos da entidade (FIWARE, 2019c).

Segundo (FIWARE, 2019c), relacionado à geração de informações de contexto, o STH possui 4 conceitos principais:

- **Resolução** ou **Período de agregação**: É o período no qual as séries de informações temporais são agrupadas.
- **Origem**: É a origem do tempo para o qual as informações de contexto da série temporal agregada se aplicam.
- **Offset**: É o deslocamento da origem para a qual as informações de contexto da série temporal agregada se aplicam.
- **Amostras**: Para determinada resolução, origem e offset, é o número de amostras, valores, eventos ou notificações disponíveis para aquele offset concreto da origem.

3.3 Cygnus

O Cygnus faz parte do projeto FIWARE, e é encarregado da persistência de dados atuando como um conector aos sistemas de armazenamento de terceiros, criando, assim uma visão do histórico dos dados. Internamente, o Cygnus é baseado no Apache Flume, que permite o design, execução da coleção de dados e persistência de agentes. Um agente é composto por um listener ou fonte encarregado do recebimento dos dados, um canal onde a fonte coloca os dados transformados em um evento Flume e um coletor que recebe eventos Flume do canal de forma a persistir os dados em um sistema de armazenamento de terceiros. Atualmente, os seguintes sistemas de armazenamentos são suportados: HDFS, MySQL, CKAN, MongoDB, STH Comet, Kafka, DynamoDB, PostgreSQL, Carto. Na arquitetura FIWARE, o Cygnus desempenha o papel de um conector entre o OCB e outros sistemas de armazenamento, oriundos ou não do FIWARE (FIWARE, 2019a).

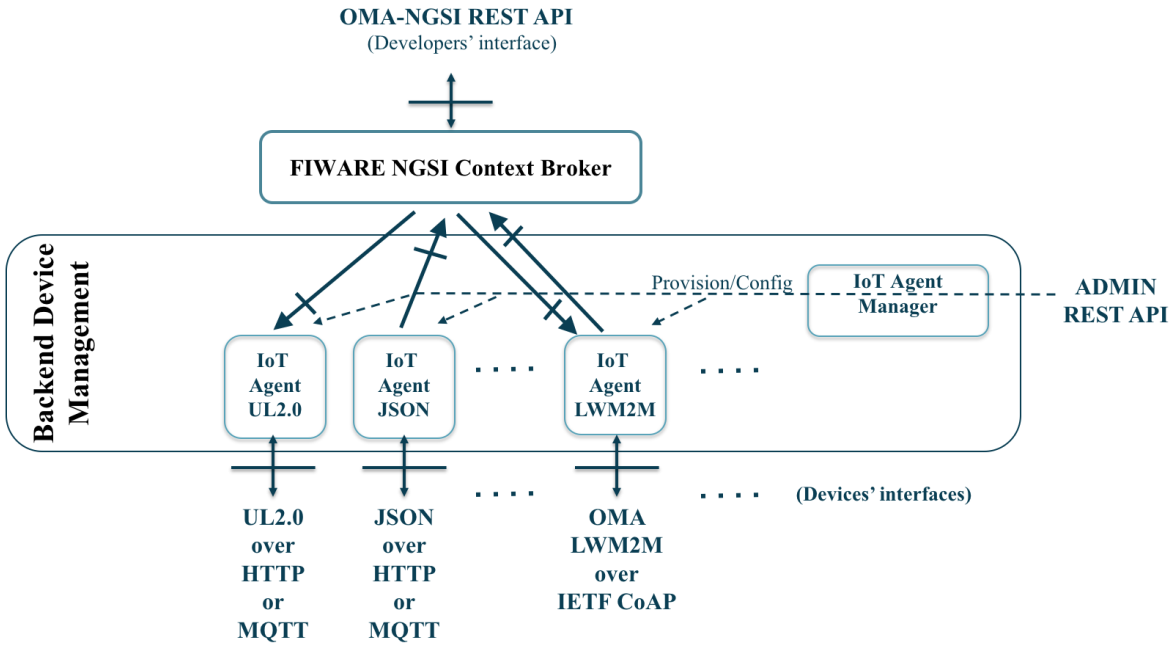
3.4 Intelligence Data Advanced Solution (IDAS)

O IDAS tem como propósito facilitar a interação com IoT e sistemas de terceiros para a coleta de informações de contexto ou disparar eventos caso ocorra uma mudança em variáveis de contexto (FIWARE, 2019b). Este oferece uma vasta gama de IoT agents que atuam traduzindo protocolos IoT específicos para o padrão NGSI (ESTRADA et al., 2016). Segundo o catálogo do IDAS (FIWARE, 2019b), estão disponíveis os seguintes IoT agents:

- **IoT Agent para JSON**: Fornece uma interface entre o sistema de mensagens HTTP / MQTT (com um *payload* JSON) e o NGSI.
- **IoT Agent para LWM2M**: Fornece uma interface entre o protocolo leve LWM2M, máquina à máquina.
- **IoT Agent para Ultralight**: Fornece uma interface entre o sistema de mensagens HTTP / MQTT (com um *payload* UltraLight 2.0) e o NGSI.
- **IoT Agent para LoRaWAN**: Fornece uma interface entre o protocolo LoRaWAN e o NGSI.
- **IoT Agent para OPC-UA**: Fornece uma interface entre o protocolo OPC Unified Architecture e o NGSI.
- **IoT Agent para Sigfox**: Fornece uma interface entre o protocolo Sigfox e o NGSI.
- **Biblioteca IoT Agent**: Fornece uma biblioteca para desenvolver seu próprio IoT agent.

Na figura 6, é mostrada a interação entre o IDAS e o OCB:

Figura 6 – Interação entre o OCB e o IDAS



Fonte: (FIWARE, 2016b)

4

Estudo de caso de aplicações IoT usando FIWARE

Neste capítulo, serão mostradas dois estudos de caso desenvolvidos como parte de dois trabalhos de conclusão de curso e que utilizam FIWARE e alguns dos seus GEs voltados para IoT. O primeiro, definido e implementado em (SILVA, 2019), trata-se de uma aplicação de sala de aula inteligente[1] que possui sensores de movimento e um ar-condicionado inteligente que gerencia seu uso de forma a otimizar o consumo energético. O segundo está definido e implementado em (LEITE, 2019) e consiste em uma aplicação voltada à um cenário que possui uma praça com lâmpadas inteligentes[2] cujo objetivo é manter a praça bem iluminada compensando lâmpadas avariadas intensificando a luminosidade das que estejam próximas. Em ambas as aplicações, foram utilizados os GEs *Orion context Broker* (OCB) e IDAS com o IoT-A do protocolo Ultralight 2.0, um dos componentes FIWARE descrito anteriormente. Além destes, foi utilizado o banco de dados MongoDB[3] e a ferramenta Docker[4]. As aplicações foram desenvolvidas utilizando a linguagem de programação Java.

No capítulo 5, é mostrado como estas duas aplicações são implementadas utilizando o *framework* orientado a objetos proposto neste trabalho, bem como é feita uma comparação utilizando parâmetros como quantidade, qualidade de código e nível de conhecimento necessário.

4.1 Iluminação inteligente de uma praça

Como foi brevemente mencionado no início do capítulo, esta aplicação consiste de uma praça que possui lâmpadas dispostas como na figura 7, as quais possuem um determinado raio de iluminação, e caso alguma sofra avaria, as demais vizinhas aumentariam seu nível de intensidade de iluminação de modo que a área continue bem iluminada, então, assim que as lâmpadas

[1] Disponível em: https://git.dcomp.ufs.br/felipematheuscs/TCC
[2] Disponível em: https://github.com/mariana-leite/TCC
[3] Disponível em: www.mongodb.com/
[4] Disponível em: www.docker.com/get-started

defeituosas voltassem a funcionar, suas vizinhas retornariam sua intensidade ao valor padrão.

Figura 7 – Representação da disposição das lâmpadas na praça inteligente

Fonte: (LEITE, 2019)

Na arquitetura da aplicação proposta, representada na figura 8 a comunicação entre a aplicação e as lâmpadas inteligentes é intermediada pelo IoT-A e o OCB. Numa primeira intermediação, a comunicação entre as lâmpadas e o OCB ocorre por meio do IoT Agent (IoT-A) via protocolo Ultralight 2.0. Uma vez que o OCB recebe apenas requisições em NGSI e a lâmpada genérica, possui um protocolo determinado pelo fabricante, o IoT-A atua como um *middleware* entre o OCB e as lâmpadas inteligentes. O objeto `Application` descrito na arquitetura realiza a criação de um *service group* fornecendo uma chave de autenticação utilizada pelos *devices* e criando uma conexão entre o IoT-A e o OCB. Além disso, são criadas as entidades que representam as lâmpadas e a praça.

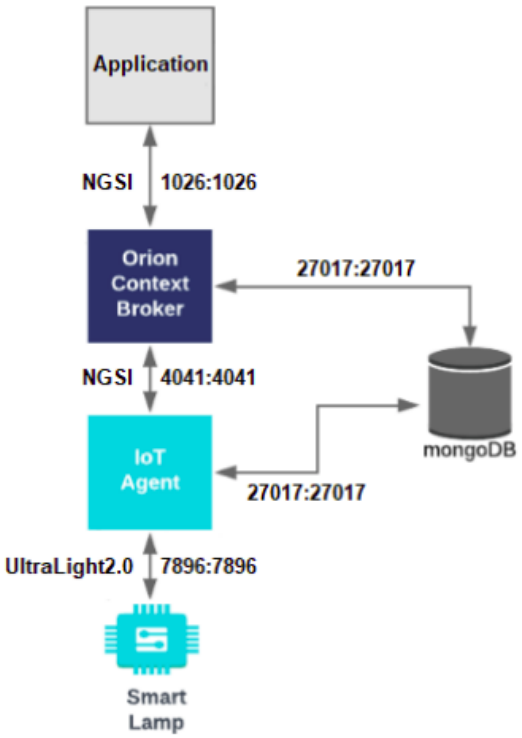
Código 7 – Criação do *service group*, *devices* e entidade praça

```
//Service Group
Services[] services = {
new Services("4jggokgpepnvsb2uv4s40d59ov",
        "http://orion:1026", "Thing", "/iot/d" )
};
ServiceGroup servicegroup = new ServiceGroup(services);
Gson gson = new Gson();
String json = gson.toJson(servicegroup);
URL url = newURL("http://localhost:4041/iot/services");
Application.postRequest(
app.getPort(),json,url,"application/json","POST");

 //Square
Attrs name = new Attrs("Praca Oliveira Belo","Text");
Attrs location = new Attrs("-10.936245,-37.061224","geo:point");
Attrs radius = new Attrs("45","Float");
Square square01 = new
   Square("urn:ngsi-ld:Square:1","Square",name,location,radius);
gson = new Gson();
json = gson.toJson(square01);
url = new URL("http://localhost:1026/v2/entities");
Application.postRequest(
    app.getPort(),json,url,"application/json","POST");

//Create Devices
Attribute[] attributes = {
    new Attribute("s","state","Text"),
    new Attribute("l","luminosity","Integer"),
    new Attribute("lo","location","geo:point"),
    new Attribute("c", "count", "Integer"),
    new Attribute("n", "number", "Integer")
};
StaticAttribute[] static_attributes = {
new StaticAttribute(
        "refSquare", "Relationship", "urn:ngsi-ld:Square:1")};
int quantity = 15;
for(int i = 0; i<quantity;i++) {
  Device[] device = {
      new Device("lamp"+(i+1),"urn:ngsi-ld:Lamp:"+(i+1),
      "Lamp",attributes,static_attributes)
  };
    CreateDevice createDevice = new CreateDevice(device);
    Gson gsonDevice = new Gson();
    String jsonDevice = gsonDevice.toJson(createDevice);
    URL urlLamp = new URL("http://localhost:4041/iot/devices");
    Application.postRequest(
    app.getPort(),jsonDevice,urlLamp,"application/json","POST");
}
```

Como pode ser visto no código 7, na instânciação do objeto `services` (linhas 1 - 5), o primeiro parâmetro é referente à chave de autenticação já citada. Após a instância do `servicegroup` (linha 6), o mesmo é serializado para JSON (linhas 7 - 8) e por fim, enviado em uma requisição POST (linhas 9 - 11). De forma análoga, a entidade `square` que representa a praça é criada, bem como os `devices`.

Para que sejam recebidas notificações sobre alterações nos estados das entidades, uma subscrição é criada no OCB indicando o conteúdo da notificação. O objeto `Application` aguarda a notificação indicando mudança de estado, que por fim, recebe o devido tratamento.

Figura 8 – Arquitetura da aplicação da iluminação inteligente em uma praça



Fonte: (LEITE, 2019)

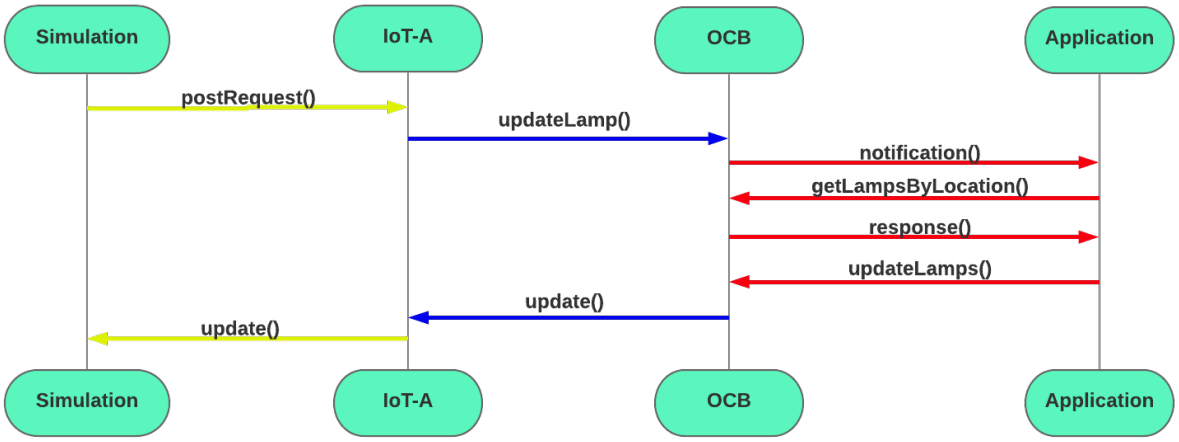
Na figura 9, é mostrado o diagrama de sequência referente ao fluxo da aplicação. Na interação dos objetos, pode ser visto que é feita uma requisição POST, indicando que dados de medições vindos dos sensores foi realizada, tais requisições são feitas por meio do IoT-A que tem como objetivo atualizar o valor das lâmpadas no OCB, assim que o valor das lâmpadas é atualizado, uma notificação é disparada, que, por sua vez, é recebida pela aplicação e por fim, faz o devido tratamento aplicando as regras de negócio definidas. Vale ressaltar que:

- As requisições entre `SensorSimulator` e `IoT-A`, como destacado em cor amarelo, são realizadas usando protocolos nativos, no caso da lâmpada simulada possuir seus próprios protocolos proprietários do fabricante;

- As requisições entre `IoT-A` e `OCB`, como destacado em azul são realizadas usando NGSI e ocorrem de forma automática de responsabilidade do próprio FIWARE;
- As requisições entre `OCB` e `Application`, como destacado em vermelho, são realizadas usando NGSI;
- Ao invés de se utilizar sensores e atuadores reais, estes foram simulados por meio do módulo `SensorSimulator`, que por sua vez, tomou a mesma implementação relatada em (LEITE, 2019).

Cabe destacar que a Figura 9 apresenta uma visão de alto nível do sistema. Em particular, do ponto de vista da `Application`, cada interação em vermelho não corresponde tão somente a uma chamada a método. Por exemplo, para a aceitação de `notification()`, `Application` atua como um servidor que recebe a notificação enviada pelo `OCB` em uma mensagem NGSI codificada em JSON por meio de um *socket*. A seguir detalhamos algumas partes relevantes das classes `Application` e `SensorSimulator` como forma de melhor entendimento e futura comparação com o *framework* proposto no Capítulo 5.

Figura 9 – Fluxo da aplicação de iluminação inteligente em uma praça



Adaptado de: (LEITE, 2019)

No código 8 a seguir, pode ser visto como é realizado o envio de medições por meio do IoT-A.

Código 8 – Código da classe `SensorSimulator` referente ao envio de informações ao IoT-A.

```
1  while(true) {
2        int randomNum = ThreadLocalRandom.current().nextInt(1,
            quantity+1);
3        boolean randomOff = ThreadLocalRandom.current().nextBoolean();
4        if(randomOff) {
5          SensorSimulator.postRequest(
6            app.getPort(),UltraLight.toStringMsg(ultralight_off),
7              new URL("http://localhost:7896/iot/d?
8              k=4jggokgpepnvsb2uv4s40d59ov
9              &i=lamp"
10             +randomNum),"text/plain","POST");
11       }
12       else {
13         SensorSimulator.postRequest(
14           app.getPort(),UltraLight.toStringMsg(ultralight_on),
15             new URL("http://localhost:7896/iot/d?
16             k=4jggokgpepnvsb2uv4s40d59ov&
17             i=lamp"
18             +randomNum),"text/plain","POST");
19       }
20
21       try{
22         Thread.sleep(1000*7);
23       }
24       catch(InterruptedException ex){
25         Thread.currentThread().interrupt();
26       }
27     }
```

O código 8 mostra como é feita a simulação dos sensores de luminosidade e o envio ao IoT-A. Inicialmente é sorteada uma lâmpada para que seja desligada ou acesa (linhas 1 - 3), caso seja simulada uma avaria na lâmpada (linhas 4 - 11), tal informação é submetida via requisição POST (linhas 5 - 10). De forma análoga a informação é submetida para o caso da lâmpada ser sorteada como acesa (linhas 12 - 19). No código 10 abaixo é mostrado o código referente à captura da notificação e aplicação de uma regra de negócio. A ação de `updateLamps()` da figura 9, é feita com os métodos `lampOff` e `lampOn`. A codificação do método `lampOff` pode ser vista no código 9:

Código 9 – Código referente ao método `lampOff`

```
1  public static void lampOff(Data notificationData, String[]
       previousState, List<ResponseFromGET> getLamps, int port) throws
       Exception {
2    if(!notificationData.getState().getValue().equals(
3    previousState[  Integer.parseInt(
4    notificationData.getNumber().getValue())
5    ])) {
6      updateLuminosity(notificationData.getId(),"0", port);
7
8      String response_string = Application.getByLocation(
9      notificationData.getLocation().getValue());
10     getLamps = Application.stringToResponseFromGET(response_string);
11
12     for(ResponseFromGET lamp : getLamps) {
13       int count_int = lamp.getCount() + 1;
14       if(lamp.getState().equals("off") &&
            !lamp.getId().equals(notificationData.getId()))
15         updateLuminosityCount(
16         lamp.getId(),"0",String.valueOf(count_int), port);
17       else if (lamp.getState().equals("on"))
18         updateLuminosityCount(
19         lamp.getId(),"3",String.valueOf(count_int), port);
20
21     }
22   }
23 }
```

O método `lampOff()` disponível no código 9, é invocado ao se detectar que uma lâmpada foi desligada, indicando um provável mal funcionamento. Neste caso, procura-se no Orion pelas lâmpadas vizinhas (linhas 8-10). Então, a luminosidade de cada uma das vizinhas é atualizada (linhas 12-19) de tal forma que, se não estiver apresentando avaria, fique mais intensa para compensar o mal funcionamento da lâmpada que provocou a execução do método.

Observe que cada lâmpada tem um atributo `count` (linha 13). Nele guarda-se a quantidade de lâmpadas vizinhas avariadas. Assim, quando a mesma volta ao seu funcionamento normal, o método `lampOn` verifica o contador de cada vizinha para determinar se a intensidade dela deve ou não retornar a ter sua luminosidade normal. O método `lampOn()`, possui uma implementação análoga ao código 9 e portanto, este não será explicitado.

Código 10 – Código referente ao recebimento de notificações e aplicação da regra de negócio no objeto `Application`

```
1  while(true) {
2    String response = app.listen();
3    gson = new Gson();
4    notification = gson.fromJson(response, NotificationResponse.class);
5    for(Data notificationData : notification.getData()) {
6
7      if(notificationData.getState().getValue().equals("off"))
8        lampOff(notificationData,previousState,app.getPort());
9
10     else if (notificationData.getState().getValue().equals("on"))
11       lampOn(notificationData,previousState,app.getPort());
12
13     previousState[Integer.parseInt(
14     notificationData.getNumber().getValue()
15     )] = notificationData.getState().getValue();
16   }
17 }
```

É interessante notar que no código 10, pode ser visto que o método `listen` do objeto `app` (linha 2), mostrado no código 11 é responsável por ouvir as notificações. Este método, realiza uma chamada bloqueante por meio de um *socket* TCP que fica bloqueado esperando uma notificação. Após o recebimento da notificação, o seu *payload* é armazenado na variável `response` e por fim, deserializado por meio do método `fromJson` do objeto Gson, oriundo da API Gson do Google (linhas 3 - 4).

Código 11 – Código referente ao método `listen`

```
1  private String listen() throws Exception {
2      String data = null;
3      String data2 = null;
4      Socket client = this.server.accept();
5
6      BufferedReader in = new BufferedReader(
7              new InputStreamReader(client.getInputStream()));
8      while ( (data = in.readLine()) != null ) {
9        data2 = data;
10     }
11     return data2;
12 }
```

Código 12 – Código referente à subscrição das entidades

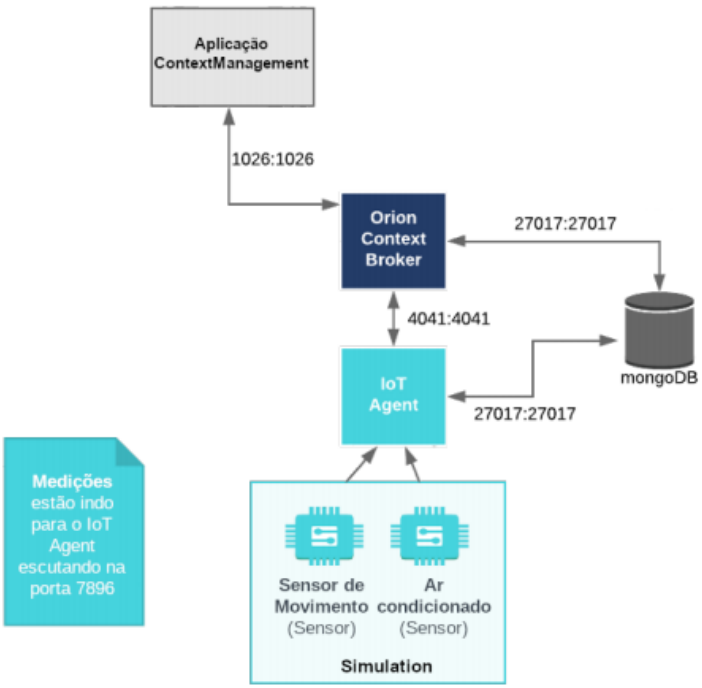
```
1  EntitiesIdPattern[] entitiesIdPattern = {new EntitiesIdPattern(".*",
       "Lamp")};
2
3  Subscription subscription = new Subscription("Mudanca de estado das
       lampadas", entitiesIdPattern, new Condition(new String[]
       {"state"}),
4         new Notification(new Http("http://172.18.1.1:"+ app.getPort()),
5         new String[] {"state","location","number","count"}) , 0);
6
7  gson = new Gson();
8  json = gson.toJson(subscription);
9
10 url = new URL("http://localhost:1026/v2/subscriptions");
11 Application.postRequest(
12          app.getPort(),json,url,"application/json","POST");
```

No código 12 acima, pode-se ver que nas linhas 1 a 5, é montado o objeto que representa o *payload* da subscrição às entidades do tipo lâmpada. Adicionalmente, nas linhas 7 e 8, o *payload* da requisição é serializado para JSON, e por fim, submetido ao orion por meio de uma requisição POST.

4.2 Sala de aula inteligente

A arquitetura da aplicação proposta, que pode ser visualizada na figura 10, consiste dos objetos `ContextManagement` e `Simulation`. Estes, possuem, respectivamente a responsabilidade de receber as notificações das subscrições realizadas no OCB e manter o controle das regras do ar-condicionado, e a simulação de eventos do sensor de presença e ar-condicionado inteligente.

Figura 10 – Arquitetura da aplicação de sala de aula inteligente



Fonte: (SILVA, 2019)

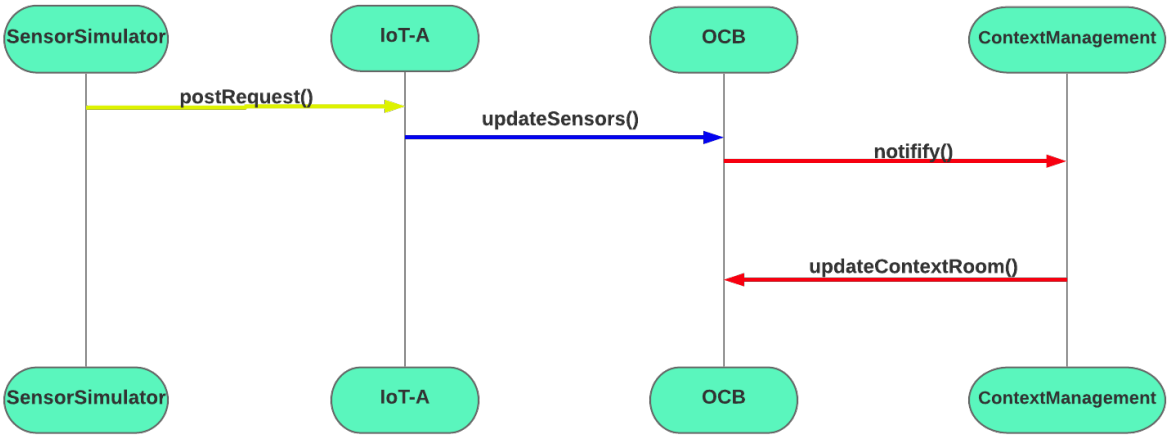
Na implementação, foi criada uma respectiva classe para cada entidade registrada no OCB. Isto, possibilitou a execução de comandos NGSI utilizando objetos bem estruturados, uma vez que ao fazer uso das bibliotecas Google HTTP Client e Gson também do Google, obteve-se uma forma de transformar objetos JSON para java e vice-versa.

Inicialmente, no `ContextManagement` é criada uma entidade do tipo sala no OCB para que as medições do ar-condicionado e do sensor de movimento sejam recebidas, é necessária a criação de um grupo de serviços para que o IoT-A realize a autenticação e tenha acesso à URL que o OCB está recebendo. Assim, por meio do arquivo de configuração, o IoT-A saberá em qual porta deverá esperar as medições dos dispositivos.

Após o registro da sala de aula no OCB e do grupo de serviços. É possível enviar as medições realizadas pelo sensor de forma direta ao IoT-A, este, é responsável pela atualização das informações por meio da interação com o OCB. Por fim, a aplicação faz as leituras e atualizações necessárias e implementa a regra de negócio.

Na figura 11, é mostrado o diagrama de sequência referente ao fluxo da aplicação. Na interação, pode ser visto que o objeto `Simulation` realiza uma requisição POST por meio do IoT-A, esta requisição oriunda do sensor, significa que o sensor de movimento detectou a entrada de pessoas na sala e seu valor foi enviado para o OCB. Assim que o valor é alterado, uma notificação é enviada ao objeto `ContextManagement`, que recebe a informação e realiza o devido tratamento utilizando as regras de negócio definidas.

Figura 11 – Fluxo da aplicação de sala de aula inteligente



Adaptado de: (SILVA, 2019)

Pode-se notar que são necessários em ambas aplicações (sobretudo na aplicação da iluminação inteligente de uma praça, a qual foi dado também detalhamento a nível de código) conceitos de programação relacionados a serialização/deserialização de objetos JSON e domínio em requisições HTTP para o uso da API. Além disso, são necessários os conceitos utilizados na plataforma FIWARE e de seus componentes, o que acaba gerando uma sobrecarga no programador, devido às múltiplas necessidades de aprendizado, e que no final, torna mais longa a curva de aprendizado e implantação do sistema desejado.

5

Framework Proposto

Desenvolver aplicações IoT por meio da plataforma FIWARE mostra-se um desafio, seja pela quantidade de conhecimentos prévios necessários em JSON, requisições HTTP e API's, ou até mesmo pela plataforma FIWARE, que por si só possui uma curva de aprendizado grande no que se refere ao entendimento e utilização de seus componentes. Tal quantidade de conhecimento prévio, acaba tornando o desenvolvimento da aplicação mais lenta, e isso, por sua vez acaba refletindo na codificação necessária, que se torna grande e com grandes blocos de código que poderiam ser abstraídos, fazendo, assim, o desenvolvedor se preocupar com aspectos técnicos que não são próprios da aplicação.

Uma solução para este problema, é a criação de um *framework* que abstraia as requisições HTTP, serialização/deserialização de arquivos JSON submetidos e/ou recuperados do OCB, dessa forma, o desenvolvedor ganha maior liberdade para aprender bem a plataforma FIWARE, dedicando para o FIWARE o tempo que seria empregado aprendendo as demais tecnologias necessárias. A solução adotada neste trabalho busca tratar as entidades do FIWARE como objetos Java, assim, para o desenvolvimento do framework, qualquer linguagem de programação orientada à objetos poderia ser adequada, no entanto para fins de comparação com sistemas relatados em (SILVA, 2019) e (LEITE, 2019), foi decidido utilizar Java como a linguagem de programação. Algumas das operações que a API NGSI[1] permite foram implementados no *framework*.

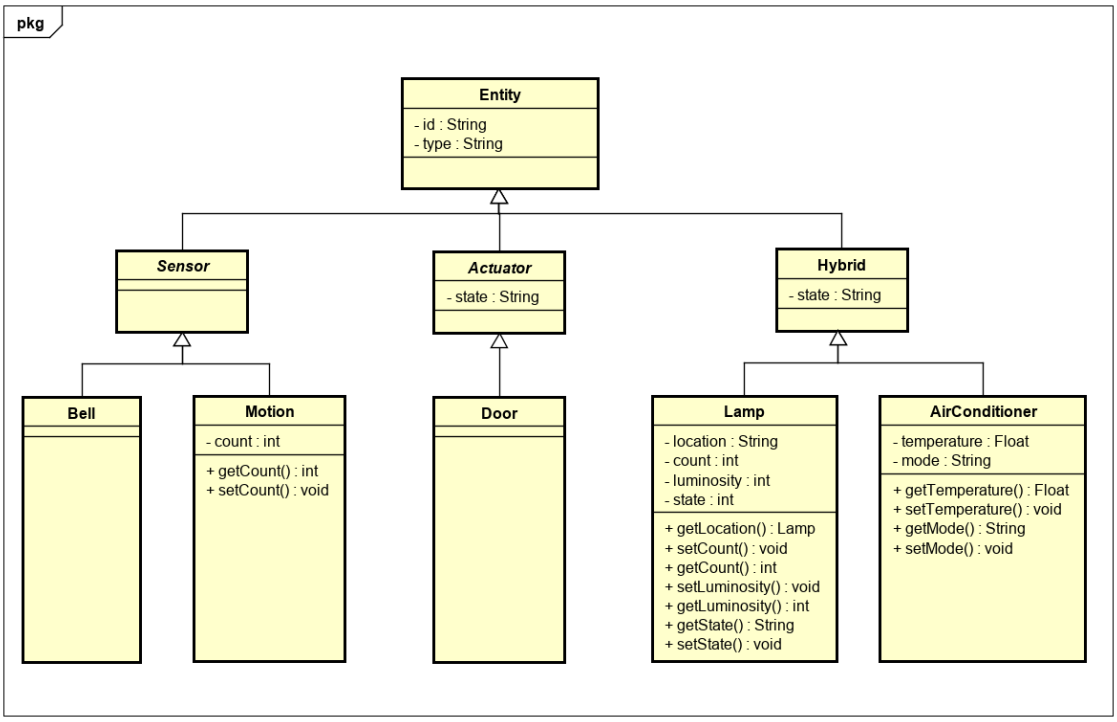
Em resumo, as funcionalidades relacionadas à consulta e atualização de entidades oferecidas pelo Orion são concentradas em um único objeto Java. Clientes deste objeto realizam consultas e/ou atualizações através de simples chamadas a métodos abstraindo o código necessário envolvido em interações REST/NGSI. Adicionalmente, o *framework* disponibiliza a funcionalidade *publish-subscribe* do Orion num estilo orientado a objetos que também abstrai a necessidade de se definir servidores apropriados para as subscrições.

[1] Disponível em: https://swagger.lab.fiware.org/

5.1 Framework Orion

As entidades, atributos e metadados do FIWARE, como definido no capítulo 3, seguem a sintaxe do NGSI no formato JSON. Na solução proposta, existem classes java que representam as entidades e atributos NGSI, os metadados não foram implementados por se tratarem de campos opcionais. No *framework*, sempre que for necessário criar uma entidade, a classe correspondente a esta deve herdar da classe `Entity`. Esta generalização parte do princípio que todo dispositivo é uma entidade, configurando assim, uma hierarquia de classes como na figura 12 abaixo:

Figura 12 – Representação da hierarquia de classes das entidades do *framework*

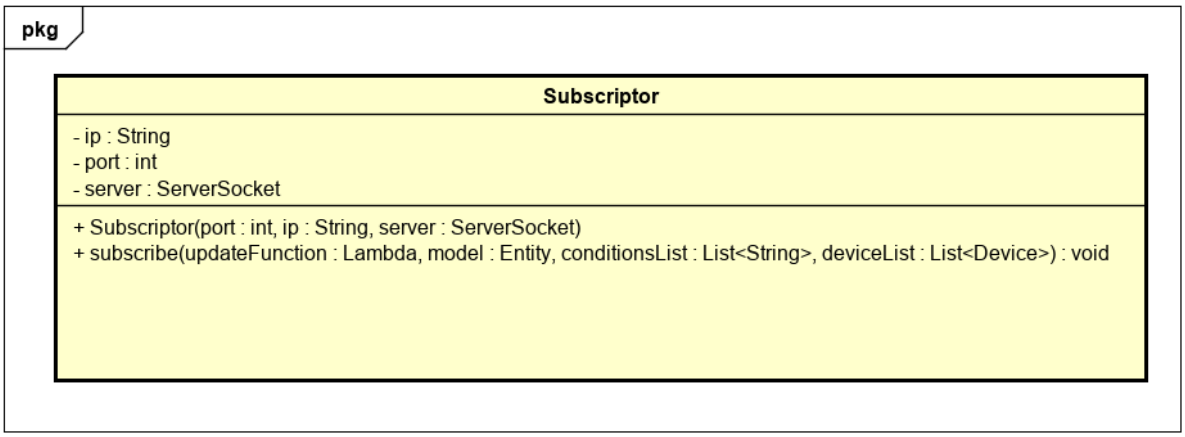


Apesar de haver a possibilidade do desenvolvedor criar suas próprias entidades, também é possível utilizar entidades já criadas e que se encontram no framework. Vale ressaltar que no diagrama da figura 12 foram denotados apenas os principais métodos de cada classe.

As principais funcionalidades do *framework* residem nas classes `Orion` e `Subscriptor`, a primeira é responsável pela implementação das operações CRUD disponíveis na API NGSI do FIWARE. Esta classe, possui dois atributos, *ip* e *port*, que significam respectivamente, o IP e porta onde a instância do OCB está sendo executada. Os métodos que a classe `Orion` implementa podem ser vistas na documentação[2]. A classe `Orion` tem como objetivo abstrair os conceitos de serialização/deserialização de arquivos no formato JSON e também abstrair as requisições HTTP feitas ao `Orion`.

[2] Disponível em: https://frameworkdocs.herokuapp.com/index.html

Figura 13 – Classe Subscriptor denotada com seus principais atributos, métodos e construtores



Pode ser observado na figura 13, os atributos, métodos e construtores da classe `Subscriptor`.

Os principais atributos da classe `Subscriptor` são:

- `ip`: Representa o ip da instância responsável por ouvir as notificações.
- `port`: Representa a porta da instância responsável por ouvir as notificações.
- `server`: Representa o servidor encarregado de aguardar as notificações.

Estes atributos são utilizados na instância de um objeto `subscriptor`, o que possibilita o uso do método `subscribe`. Os demais atributos desta classe, bem como as suas informações estão disponíveis na documentação[3].

Como forma de exemplificar, considere o seguinte construtor: `Subscriptor(int port, String ip, ServerSocket server)`, Tal instância permite que se faça o correto uso do método `subscribe`.

A classe `Subscriptor` tem como objetivo facilitar ainda mais o trabalho do desenvolvedor. Esta, traz a funcionalidade de cadastrar registros de subscrição por atualizações em entidades do orion e aguardar por notificações de alteração dos atributos da entidade em questão. O método `subscribe()`, provido por esta classe, permite cadastrar um *callback* a ser executado quando há uma mudança de estado em uma das entidades subscritas. O método aceita como argumento a função de *callback* assim como as entidades que disparam a notificação junto com as condições de disparo. Assim que uma notificação é recebida, o conteúdo da mesma é automaticamente deserializado e por fim, o *callback* correspondente é executado.

Na figura 14, pode ser vista a interação entre o desenvolvedor e o *framework*. É possível notar que ao realizar a chamada a um dos métodos do *framework*, em *background*, ou seja, de

[3] Disponível em: https://frameworkdocs.herokuapp.com/index.html

modo abstrato ao programador, é feita a serialização/deserialização do objeto passado como parâmetro de forma a obter o *payload* da requisição HTTP, assim, fazendo uso desse *payload*, a requisição HTTP feita ao OCB é realizada. Esta requisição está diretamente relacionada ao método que o programador está chamando, que pode ou não possuir retorno de valores.

Figura 14 – Interação entre o código fonte do desenvolvedor e o *framework*

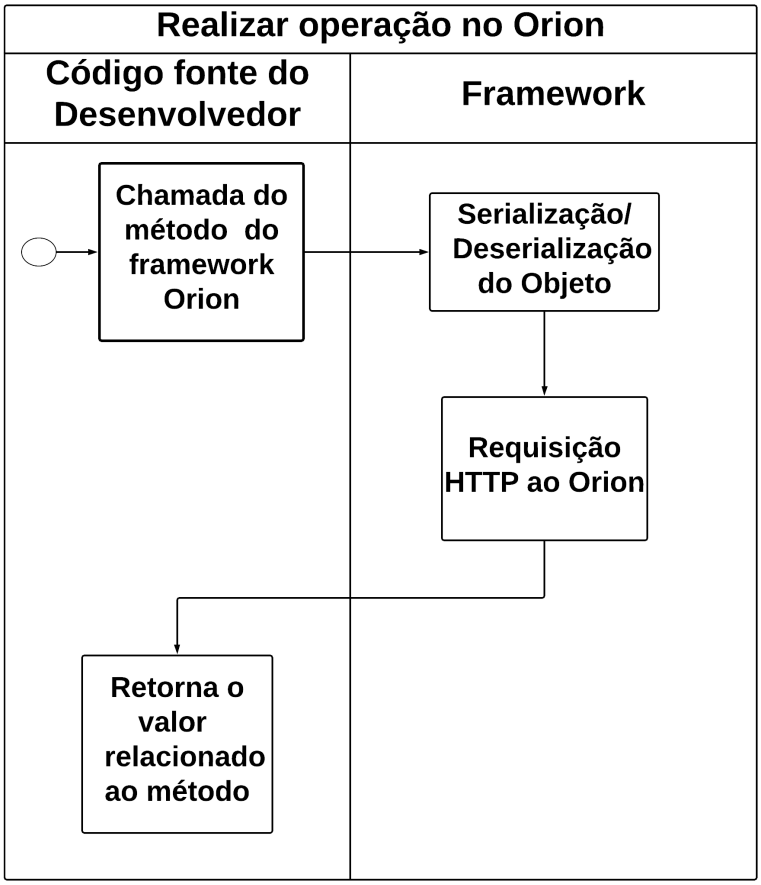
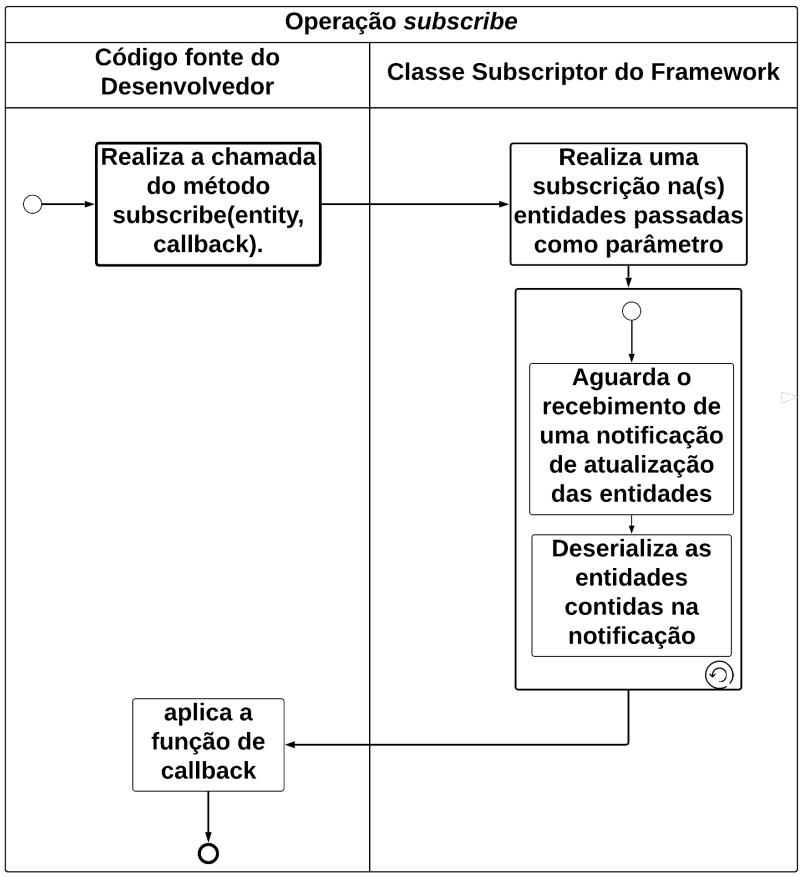


Figura 15 – Interação entre o código fonte do desenvolvedor e o *framework* no método `subscribe()`



Uma interação interessante no *framework* a se explicitar, se dá na operação ressaltada na figura 15. Aqui, o desenvolvedor possui a responsabilidade de realizar subscrições passando a correspondente função de *callback*, o *framework* possui uma *thread* específica que realiza as subscrições no `Orion` e aguarda o recebimento de notificações, assim que uma estiver disponível, as entidades necessárias são deserializadas para aplicação da função de *callback*.

Neste trabalho foram consideradas apenas algumas das principais operações que envolvem *entities, attributes e subscriptions*. Em resumo, estas são:

- `createEntity`: Método responsável pela criação de entidades no *broker*;
- `listEntities`: Método responsável por recuperar entidades do *broker*;
- `updateAttributeValue`: Método responsável pela atualização do valor de uma entidade no *broker*;
- `createSubscriptions`: Método responsável pela criação de uma subscrição no broker;
- `subscribe()`: Método responsável pela criação de uma subscrição e aguardar notificações com respectivas atualizações;

Ao leitor que se interessar, o *framework* proposto possui uma documentação[4] com a listagem das operações disponíveis, bem como os seus devidos detalhes.

5.2 Estudos de caso: Iluminação inteligente de uma praça e sala de aula inteligente.

Como prova de conceito, as aplicações estudadas no capítulo 4 foram reescritas utilizando o *framework* desenvolvido. A seguir apresentamos trechos de código da aplicação de iluminação da praça. Na codificação 13 pode ser visto o trecho de código relativo à subscrição por alterações no estado das lâmpadas (linha 1). A subscrição é feita com uma chamada ao método `subscribe()` passando como argumento o método *callback* `updateEntitiy` (linhas 5 - 13) que será invocado quando ocorrer alguma mudança no estado de alguma das lâmpadas.

Código 13 – Código 10 reescrito com o *framework*

```
1  subscriptor.subscribe(en -> updateEntity((Lamp) en), new Lamp(),
       conditionsList, deviceList);
2
3  ...
4
5  public static Lamp updateEntity(Lamp l) {
6
7    if (l.getState().getValue().equals("off")){
8      lampOffFramework(l, previousState, orion);
9    }else {
10     lampOnFramework(l, previousState, orion);
11   }
12   previousState[Integer.parseInt(l.getNumber().getValue())] =
         l.getState().getValue();
13 }
```

Note que, na codificação 13, o método `updateEntity` é enviado como parâmetro do `subscribe()` por meio de uma expressão lambda, por fim, o `subscribe()` aguarda o recebimento da notificação e aplica o método no conteúdo da notificação. Por fim, pode-se notar que as notificações HTTP e o tratamento de objetos JSON foi abstraído, Adicionalmente o código referente à escuta da notificação exibido no código 11 por meio de um *socket* não se faz mais necessário, pois o *framework* faz tudo isto de forma transparente para o programador.

[4] Disponível em: https://frameworkdocs.herokuapp.com/index.html

Código 14 – Código 9 reescrito com o *framework*

```
1
2  public static void lampOffFramework(Lamp lamp, String[]
       previousState) {
3
4      if (!lamp.getState().equals(previousState[lamp.getNumber()])){
5
6          lamp.setLuminosity(0, iota);
7        List<Lamp> lampList = lamp.getByLocation(orion);
8
9        for (Lamp lamps : lampList) {
10             int count_int = lamps.getCount() + 1;
11         if (lamps.getState().equals("off") &&
               !lamps.getId().equals(lamp.getId())){
12           lamps.setLuminosity(0, iota);
13           lamps.setCount(count_int, iota);
14         } else if (lamps.getState().equals("on")) {
15           lamps.setLuminosity(3, iota);
16           lamps.setCount(2, iota);
17         }
18       }
19     }
20 }
```

O código 14, implementa a regra relacionada ao caso de uma lâmpada estar avariada, aumentando assim, a luminosidade de suas lâmpadas vizinhas. Nesta codificação, Fazendo uma comparação com o código do mesmo sistema implementado sem o *framework*, o código 14, aqui exposto, corresponde ao código 9 apresentado no capítulo anterior. Particularmente, note que há uma abstração das linhas 8 a 10 no código 9, eliminando a necessidade de montagem de requisições e deserialização de entidades o que simplifica o trabalho do programador.

Uma implementação do sistema de ar-condicionado inteligente relatado no capítulo 4 usando o framework também foi feita e encontra-se disponível no código fonte da aplicação [5].

[5] https://bitbucket.org/RomarioBispo/orion

6
Conclusão

Este trabalho propiciou nosso entendimento de IoT, cidades inteligentes e, particularmente, de como desenvolver aplicações IoT para cidades inteligentes usando componentes do FIWARE. Com isso, foi possível perceber a necessidade da redução de trabalho no desenvolvimento de aplicações FIWARE, bem como a redução na curva de aprendizado. Assim, nós definimos e prototipamos um *framework* tentando atingir simplicidade para o programador.

Para se atingir o desenvolvimento do *framework*, foi realizado um estudo de caso, envolvendo as duas aplicações desenvolvidas em Java citadas no capítulo 4, neste estudo, constata-se que tais aplicações possuem um código poluído com as interações realizadas com a API do NGSI, uma vez que é necessária a serialização/deserialização de objetos JSON. Estas interações comprometem fatores de qualidade, como legibilidade e modularidade do código, isto é, acaba implicando em um código de difícil manutenção.

Após o estudo de caso, foram detectados e definidas as principais funcionalidades que o *framework* possuiria. Tomando como base as operações da API do NGSI, o *framework* foi desenvolvido utilizando a linguagem de programação Java. No entanto, ainda com base na análise das necessidades, foi desenvolvido o módulo *subscriptor*.

Foi observado, que com o uso do framework alcançou-se melhoria na qualidade do código, se comparado com o código das aplicações em (LEITE, 2019) e (SILVA, 2019). Tal melhoria abrange legibilidade e modularidade. Isto é uma consequência do framework abstrair detalhes comuns da programação com bibliotecas baseadas em ReST.

Em concordância com os estudos realizados, é notável que a plataforma FIWARE dispõe de um grande potencial futuro, os temas IoT e cidades inteligentes cada vez mais tem se tornado recorrentes em pesquisas científicas, e tanto a indústria, quanto a academia têm demonstrado interesse nesse assunto.

Em trabalhos futuros, pretende-se estender o protótipo do *framework*, trazendo a im-

plementação de mais funcionalidades e componentes do FIWARE, bem como a melhoria das *features* hoje disponíveis e a criação de mais entidades já integradas ao FIWARE.

Referências

AHLGREN, B.; HIDELL, M.; NGAI, E. C.-H. Internet of things for smart cities: Interoperability and open data. *IEEE Internet Computing*, IEEE, v. 20, n. 6, p. 52–56, 2016. Citado na página 13.

ARASTEH, H. et al. Iot-based smart cities: a survey. In: IEEE. *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*. [S.l.], 2016. p. 1–6. Citado na página 13.

ASHTON, K. et al. That 'internet of things' thing. *RFID journal*, v. 22, n. 7, p. 97–114, 2009. Citado na página 16.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer networks*, Elsevier, v. 54, n. 15, p. 2787–2805, 2010. Citado 2 vezes nas páginas 26 e 27.

ATZORI, L.; IERA, A.; MORABITO, G. Understanding the internet of things: definition, potentials, and societal role of a fast evolving paradigm. *Ad Hoc Networks*, Elsevier, v. 56, p. 122–140, 2017. Citado na página 16.

BALAKRISHNA, C. Enabling technologies for smart city services and applications. In: IEEE. *2012 sixth international conference on next generation mobile applications, services and technologies*. [S.l.], 2012. p. 223–227. Citado na página 25.

BAWANY, N. Z.; SHAMSI, J. A. Smart city architecture: Vision and challenges. *International Journal of Advanced Computer Science and Applications*, v. 6, n. 11, p. 246–255, 2015. Citado na página 25.

BOTTA, A. et al. Integration of cloud computing and internet of things: a survey. *Future generation computer systems*, Elsevier, v. 56, p. 684–700, 2016. Citado na página 13.

BOULTON, A.; BRUNN, S. D.; DEVRIENDT, L. 18 cyberinfrastructures and 'smart'world cities: physical, human and soft infrastructures. *International handbook of globalization and world cities*, p. 198, 2011. Citado na página 23.

CHEN, R.; GUO, J.; BAO, F. Trust management for soa-based iot and its application to service composition. *IEEE Transactions on Services Computing*, IEEE, v. 9, n. 3, p. 482–495, 2016. Citado na página 18.

CHENG, B. et al. Situation-aware iot service coordination using the event-driven soa paradigm. *IEEE Transactions on Network and Service Management*, IEEE, v. 13, n. 2, p. 349–361, 2016. Citado 2 vezes nas páginas 17 e 18.

CHOURABI, H. et al. Understanding smart cities: An integrative framework. In: IEEE. *2012 45th Hawaii international conference on system sciences*. [S.l.], 2012. p. 2289–2297. Citado na página 23.

COLA, S. D. et al. A heterogeneous approach for developing applications with fiware ges. In: SPRINGER. *European Conference on Service-Oriented and Cloud Computing*. [S.l.], 2015. p. 65–79. Citado na página 28.

COULOURIS, G. et al. *Distributed Systems: Concepts and Design. 5th*. [S.l.]: USA: Addison-Wesley Publishing Company, 2011. Citado 2 vezes nas páginas 29 e 30.

CUNHA, M. A. et al. *Smart cities: transformação digital de cidades*. [S.l.]: Programa Gestão Pública e Cidadania, 2016. Citado na página 23.

ESTRADA, H. et al. D3. 2: Smartsdk iot and data management enablers. 2016. Citado na página 35.

FAZIO, M. et al. Exploiting the fiware cloud platform to develop a remote patient monitoring system. In: IEEE. *2015 IEEE Symposium on Computers and Communication (ISCC)*. [S.l.], 2015. p. 264–270. Citado na página 28.

FERNÁNDEZ, P. et al. Smartport: a platform for sensor data monitoring in a seaport based on fiware. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 16, n. 3, p. 417, 2016. Citado 2 vezes nas páginas 29 e 30.

FIELDING, R. T.; TAYLOR, R. N. *Architectural styles and the design of network-based software architectures*. [S.l.]: University of California, Irvine Doctoral dissertation, 2000. v. 7. Citado na página 18.

FIWARE. *Catálogo FIWARE*. 2016. Disponível em: <https://www.fiware.org/developers/catalogue/>. Acesso em: 06 agosto 2019. Citado na página 29.

FIWARE. *wiki IDAS*. 2016. Disponível em: <https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.IoT.Backend.DeviceManagement>. Acesso em: 21 agosto 2019. Citado na página 36.

FIWARE. *Documentação Cygnus*. 2019. Disponível em: <https://fiware-cygnus.readthedocs.io/en/latest/>. Acesso em: 07 maio 2019. Citado na página 35.

FIWARE. *Documentação IDAS*. 2019. Disponível em: <https://www.fiware.org/developers/catalogue/>. Acesso em: 09 maio 2019. Citado na página 35.

FIWARE. *Documentação STH*. 2019. Disponível em: <https://fiware-sth-comet.readthedocs.io/en/latest/>. Acesso em: 07 maio 2019. Citado na página 34.

FIWARE FOUNDATION. *API FIWARE*. 2018. Disponível em: <http://fiware.github.io/specifications/ngsiv2/stable/>. Acesso em: 21 agosto 2019. Citado na página 19.

FIWARE-NGSI. *FIWARE NGSIV2*. 2018. Disponível em: <http://fiware.github.io/specifications/ngsiv2/stable/>. Acesso em: 07 agosto 2019. Citado 2 vezes nas páginas 30 e 31.

FOSTER, A. Messaging technologies for the industrial internet and the internet of things. *PrismTech Whitepaper*, 2015. Citado na página 21.

GIFFINGER, R.; PICHLER-MILANOVIĆ, N. *Smart cities: Ranking of European medium-sized cities*. [S.l.]: Centre of Regional Science, Vienna University of Technology, 2007. Citado na página 23.

HAN, N. S. *Semantic service provisioning for 6LoWPAN: powering internet of things applications on Web*. Tese (Doutorado) — Institut National des Télécommunications, 2015. Citado na página 21.

HAN, S. N.; CRESPI, N. Semantic service provisioning for smart objects: Integrating iot applications into the web. *Future Generation Computer Systems*, Elsevier, v. 76, p. 180–197, 2017. Citado na página 13.

HARRISON, C. et al. Foundations for smarter cities. *IBM Journal of research and development*, IBM, v. 54, n. 4, p. 1–16, 2010. Citado na página 23.

HOLLANDS, R. G. Will the real smart city please stand up? intelligent, progressive or entrepreneurial? *City*, Taylor & Francis, v. 12, n. 3, p. 303–320, 2008. Citado na página 23.

HOLLER, J. et al. *From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence*. 1st. ed. Orlando, FL, USA: Academic Press, Inc., 2014. ISBN 012407684X, 9780124076846. Citado 2 vezes nas páginas 16 e 17.

json.org. *Introdução ao JSON*. 2001. Disponível em: <http://www.json.org/>. Acesso em: 21 agosto 2019. Citado na página 18.

KHAN, R. et al. Future internet: the internet of things architecture, possible applications and key challenges. In: IEEE. *2012 10th international conference on frontiers of information technology*. [S.l.], 2012. p. 257–260. Citado na página 22.

KIM, T.-h.; RAMOS, C.; MOHAMMED, S. *Smart city and IoT*. [S.l.]: Elsevier, 2017. Citado na página 23.

KON, F.; SANTANA, E. F. Z. Cidades inteligentes: Conceitos, plataformas e desafios. *Jornadas de Atualização em Informática*, p. 17, 2016. Citado 2 vezes nas páginas 24 e 25.

KRAIJAK, S.; TUWANUT, P. A survey on internet of things architecture, protocols, possible applications, security, privacy, real-world implementation and future trends. In: IEEE. *2015 IEEE 16th International Conference on Communication Technology (ICCT)*. [S.l.], 2015. p. 26–31. Citado 2 vezes nas páginas 20 e 21.

LEITE, M. M. d. A. Estudo sobre iot, cidades inteligentes e fiware. DCOMP-Departamento de Computação–Engenharia de Computação–São Cristóvão . . . , 2019. Citado 8 vezes nas páginas 14, 15, 37, 38, 40, 41, 48 e 55.

LUZURIAGA, J. E. et al. A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks. In: IEEE. *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*. [S.l.], 2015. p. 931–936. Citado na página 21.

MARSH, G. et al. Scaling advanced message queuing protocol (amqp) architecture with broker federation and infiniband. *Ohio State University, Tech. Rep. OSU-CISRC-5/09-TR17*, p. 38, 2008. Citado na página 21.

MARTÍNEZ-BALLESTÉ, A.; PÉREZ-MARTÍNEZ, P. A.; SOLANAS, A. The pursuit of citizens' privacy: a privacy-aware smart city is possible. *IEEE Communications Magazine*, IEEE, v. 51, n. 6, p. 136–141, 2013. Citado na página 25.

MASSE, M. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. [S.l.]: "O'Reilly Media, Inc.", 2011. Citado na página 18.

MIORANDI, D. et al. Internet of things: Vision, applications and research challenges. *Ad hoc networks*, Elsevier, v. 10, n. 7, p. 1497–1516, 2012. Citado na página 17.

NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In: IEEE. *2017 IEEE international systems engineering symposium (ISSE)*. [S.l.], 2017. p. 1–7. Citado 2 vezes nas páginas 20 e 21.

NAPHADE, M. et al. Smarter cities and their innovation challenges. *Computer*, IEEE, v. 44, n. 6, p. 32–39, 2011. Citado na página 26.

ONG, S. P. et al. The materials application programming interface (api): A simple, flexible and efficient api for materials data based on representational state transfer (rest) principles. *Computational Materials Science*, Elsevier, v. 97, p. 209–215, 2015. Citado na página 18.

PALATTELLA, M. R. et al. Standardized protocol stack for the internet of (important) things. *IEEE communications surveys & tutorials*, IEEE, v. 15, n. 3, p. 1389–1406, 2013. Citado 2 vezes nas páginas 20 e 21.

PELLICER, S. et al. A global perspective of smart cities: A survey. In: IEEE. *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. [S.l.], 2013. p. 439–444. Citado na página 23.

RAHMAN, L. F.; OZCELEBI, T.; LUKKIEN, J. J. Choosing your iot programming framework: Architectural aspects. In: IEEE. *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*. [S.l.], 2016. p. 293–300. Citado na página 19.

SANTANA, E. F. Z. et al. Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. *ACM Computing Surveys (CSUR)*, ACM, v. 50, n. 6, p. 78, 2018. Citado na página 25.

SILVA, F. M. C. d. Estudo de componentes de fiware para iot e cidades inteligentes. DCOMP-Departamento de Computação–Engenharia de Computação–São Cristóvão . . . , 2019. Citado 7 vezes nas páginas 14, 15, 37, 46, 47, 48 e 55.

SKARMETA, A. F.; HERNANDEZ-RAMOS, J. L.; MORENO, M. V. A decentralized approach for security and privacy challenges in the internet of things. In: IEEE. *2014 IEEE world forum on Internet of Things (WF-IoT)*. [S.l.], 2014. p. 67–72. Citado na página 19.

VILAMOVSKA, A. et al. Rfid application in healthcare–scoping and identifying areas for rfid deployment in healthcare delivery. *RAND Europe, February*, 2009. Citado na página 26.

WELBOURNE, E. et al. Building the internet of things using rfid: the rfid ecosystem experience. *IEEE Internet computing*, IEEE, v. 13, n. 3, p. 48–55, 2009. Citado na página 26.

WHITMORE, A.; AGARWAL, A.; XU, L. D. The internet of things—a survey of topics and trends. *Information Systems Frontiers*, Springer, v. 17, n. 2, p. 261–274, 2015. Citado na página 26.

XU, L. D.; HE, W.; LI, S. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, IEEE, v. 10, n. 4, p. 2233–2243, 2014. Citado 2 vezes nas páginas 17 e 18.

YIN, C. et al. A literature survey on smart cities. *Science China Information Sciences*, Springer, v. 58, n. 10, p. 1–18, 2015. Citado na página 24.

ZANELLA, A. et al. Internet of things for smart cities. *IEEE Internet of Things journal*, IEEE, v. 1, n. 1, p. 22–32, 2014. Citado 3 vezes nas páginas 13, 18 e 23.

ZHANG, K. et al. Security and privacy in smart city applications: Challenges and solutions. *IEEE Communications Magazine*, IEEE, v. 55, n. 1, p. 122–129, 2017. Citado na página 25.

ZOONEN, L. V. Privacy concerns in smart cities. *Government Information Quarterly*, Elsevier, v. 33, n. 3, p. 472–480, 2016. Citado na página 24.